UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE GRADO

DISEÑO DE UNA INTERFAZ PARA UNA APLICACIÓN ASÍ COMO EL CONTROL TÁCTIL DE LA MISMA Y DISEÑO DE LAS ESTRUCTURAS DE DATOS NECESARIAS PARA EL FUNCIONAMIENTO DE LA APLICACIÓN

Gonzalo López Gómez

Julio 2014

DISEÑO DE UNA INTERFAZ PARA UNA APLICACIÓN ASÍ COMO EL CONTROL TÁCTIL DE LA MISMA Y DISEÑO DE LAS ESTRUCTURAS DE DATOS NECESARIAS PARA EL FUNCIONAMIENTO DE LA APLICACIÓN

AUTOR: Gonzalo López Gómez
TUTOR: Simone Santini

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio 2014

Resumen

Parece increíble pensar que, no hace mucho tiempo, no era nada común tener un ordenador en casa, y mucho menos acceso a internet. Y es que los grandes avances que surgen día a día en este campo siguen asombrando a todo el mundo. Antes, si querías utilizar internet, tenías que prescindir del teléfono fijo durante un tiempo, pero ahora, en tan solo unos años, la gente camina por la calle hablando con teléfonos móviles mientras descargan todo tipo de contenido de la red.

Es normal que las grandes compañías se muestren muy interesadas en este mercado y que cada vez sean más las aplicaciones y herramientas que se pueden utilizar desde estos dispositivos. Uno de los sectores más beneficiados de este rápido proceso de mejora son los juegos. Los grandes avances en el rendimiento y la presentación de las aplicaciones han fomentado enormemente el desarrollo de nuevos juegos y la remasterización de otros.

El objetivo principal que se persigue en este proyecto es el desarrollo de un juego para cualquier tipo de dispositivo portátil que utilice Android. Lo que resulta más interesante del juego y de la aplicación, es que pretenden combinar un juego de mesa, como los de toda la vida, con las funcionalidades de estas nuevas tecnologías. Mientras que se mantienen los elementos físicos de los juegos de mesa, la aplicación se encarga de gestionar gran parte de la jugabilidad, evitando así problemas con el entendimiento de las reglas y haciéndolo mucho más entretenido y cómodo para los jugadores.

El siguiente documento recoge, además de una breve descripción del juego desarrollado y de los objetivos y retos que se perseguían con la realización del proyecto, toda la información necesaria para comprender el trabajo realizado en cada una de las fases que se han seguido, desde las etapas previas de estudio, hasta casi la conclusión del proyecto.

Palabras clave

Android, Móvil, Aplicación, Juego de mesa, Online, Multijugador

Abstract

It seems almost incredible to believe that, not so long ago, it was uncommon to have a personal computer at home, and even less, an internet connection, but every single day, great advances arise and continue to amaze everyone. Back then, it was impossible to use the internet and the landline at the same time, but now, in only a few years, people walk down the street talking on mobile phones while downloading all kinds of content from the network.

It is only natural for large companies to become very interested in this market and that more and more applications and tools, which can be used in these devices, are released. One of the most benefited sectors from this fast improvement is games. Major progresses in the performance and interface of applications have greatly promoted the development of new games and some older ones.

The main objective pursued in this project is to develop a game for any kind of mobile device that uses Android. What is most interesting about the game and the application is that it intends to combine a board game with the functionalities of these new technologies. While the physical elements of the board game remain, the application is responsible for managing many aspects of the game play, thus it avoids problems with the understanding of the rules and makes it much easier, fun and comfortable for the players.

The following document contains, along with a brief description of the developed game and the goals and challenges pursued with the project, all the information necessary to understand the work done in each of the phases that have been followed, from the previous stages of study, to almost the end of the project.

Index Terms

Android, Mobile, Application, Game, Online, Multiplayer

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

En la actualidad y probablemente a medida que pasen los años con mayor impacto, la tecnología móvil continúa adquiriendo una grandísima importancia en nuestras vidas. Una prueba de ello son los *Smartphone* [2], que han sido capaces de convertirse en un elemento casi imprescindible tanto en el ámbito personal, como en el profesional. Cada vez son más las tareas que se pueden realizar con ellos y su rápida adaptación a cualquier tipo de necesidades nos lleva a exigir día a día nuevos retos a sus desarrolladores. Obviamente, y como pasa con todo, aunque también tiene sus inconvenientes, queda bastante claro que las virtudes de estas nuevas tecnologías superan con creces cualquier desventaja que puedan tener.

A medida que han ido transcurriendo los años, y con los grandes y rápidos avances que se producen en este campo, han sido muchas las compañías que han decido intentar sacar provecho de esta situación. Un buen ejemplo de ello son los juegos de mesa, que a partir de aplicaciones para este tipo de terminales han conseguido que sus usuarios puedan seguir disfrutando de los juegos clásicos de toda la vida, y que tanto les gustaban, de una forma mucho más cómoda. Algunas aplicaciones famosas que han seguido este proceso han sido el *Scrabble* con su versión móvil *Apalabrados*, el Trivial con el *Triviados*, el *Pictionary* con el *DrawSomething* y muchas otras que, aunque no hayan tenido una aceptación tan grande, siguen resultando muy buenas.

Precisamente esto último es lo que ha impulsado la realización de este proyecto. Como combinar las maravillosas ventajas de los cada vez mejores dispositivos móviles, con los juegos de mesa de toda la vida, manteniendo la idea de lo que son los juegos de mesa. La idea principal, más que combinar las dos plataformas, consiste en integrar las funciones de los dispositivos móviles en un juego real de mesa.

1.1 Objetivos y retos

Uno de los mayores, y probablemente de los más gratificantes retos que un desarrollador de software puede afrontar, es la realización completa de un proyecto, de inicio a fin, participando en todas sus etapas y poniendo a prueba todos sus conocimientos y habilidades con el único fin de conseguir terminarlo con éxito y hacerlo suyo. En línea con lo anterior, los proyectos grandes implican un mayor grado de estimación de tiempos y recursos que dificultan todavía más la tarea.

Otro de los retos importantes para la realización de este proyecto es aprender a programar aplicaciones para dispositivos Android. Aunque tener una base solida de Java, XML y SQL, facilitará bastante el trabajo, se deberá comprender como funciona y como aprovechar las características y facilidades que brinda Android.

Mejorar como profesionales es uno de los retos que indudablemente se persiguen con la realización del trabajo de fin de grado, es un punto y final a la carrera y se ha preferido realizar un proyecto propio y no elegir uno de los ofertados por la universidad. Con esto se persigue fomentar el trabajo independiente y dar mayor peso a la toma de decisiones en el proyecto.

Por último, como ha pasado con todos los trabajos realizados en la carrera, se encuentra el objetivo de aprender cosas nuevas que no se hayan enseñado, porque las cosas no se aprenden realmente hasta que no las tiene que hacer un mismo.

1.2 Introducción a la aplicación

A continuación se explicará en qué va a consistir muy por encima el juego que se quiere desarrollar para este proyecto, y como se combinará con la aplicación móvil que se va a desarrollar.

Historia:

En el año 1990, a causa de la contaminación, el agua es portadora de un virus que contagia a todos los seres vivos del planeta. Las grandes potencias del mundo mandan construir distintos laboratorios respaldados por las fuerzas militares, con la intención de investigar acerca de esta plaga. Deciden ocultar esta información a la población ya que los síntomas de esta nueva epidemia no parecen empezar a tener efecto hasta pasados 10-15 años.

Transcurridos 10 años se produce lo que se conoce como *"el efecto 2000"*, y todos los dispositivos electrónicos se desconfiguran. Debido a esto se producen numerosos ataques informáticos a los distintos servicios de inteligencia mundiales y sale a la luz el secreto del virus. Las revueltas comienzan a ser constantes y empiezan a formarse grupos anti-sistema. En algunos de los centros de investigación se descubre una forma de ralentizar el efecto del virus mediante un suero, pero este solo se suministrará a aquellos que se sometan a una cirugía experimental. El gobierno insiste en describirlo como un nuevo avance evolutivo necesario para el ser humano, pero en realidad, su único objetivo es aumentar el control del gobierno sobre la población y evitar nuevas revueltas. Esta nueva cirugía consiste en la implantación de un dispositivo *IDlog* en el antebrazo que estará conectado al cerebro. Aquellas personas sometidas a la cirugía serían trasladadas a búnkeres construidos por todo

el mundo, donde se les suministraría agua potable y se le protegería de las distintas amenazas que fueran surgiendo en el exterior (contacto directo con el virus, saqueadores, grupos de la resistencia y animales).

Durante el año 2014, uno de los centros de investigación más importantes manda una señal de radio anunciando que han encontrado una cura definitiva para el virus, pero cuando intentan volver a contactar con ellos la señal se corta. Cinco de los búnkeres más cercanos mandan a una persona para que intente conseguir la cura, pero… ¿con qué propósito?

Objetivos:

El objetivo principal del juego es conseguir la cura para tu búnker antes que el resto. Para ello, los jugadores podrán decidir si cooperar, luchar o ignorarse entre ellos. Gracias al dispositivo que todos los miembros de los búnkeres tienen implantados, los jugadores pueden monitorizar sus habilidades, su inventario y su salud y elegir como quieren enfocar su evolución dentro del juego. Los 5 grupos de habilidades que existen son conflicto, creación, diálogo, exploración e informática y dependiendo de ellas el jugador será capaz de realizar unas acciones más específicas. Durante la partida es posible que los objetivos de los jugadores cambien en función de cómo jueguen, por ejemplo, si un jugador pierde, en vez de dejar de jugar, deberá intentar que ninguno de sus contrincantes consiga la cura y gane el juego.

Aplicación:

La aplicación será la encargada de gestionar las habilidades, el inventario, la salud, los turnos, los conflictos y las acciones que realicen los jugadores. Todas las cartas físicas del juego dispondrán de un *código QR* que se podrá escanear desde el juego y permitirá añadir objetos y resolver conflictos y amenazas. Gestionará las acciones que cada jugador puede realizar durante su turno y fuera de él y comunicará todos los dispositivos de los jugadores que participan en la partida para realizar los intercambios de información necesarios. También tramitará toda la información vigente de cada partida para permitir crear, unirse, guardar y cargar cuando se desee. Con esto, aparte de evitar que los jugadores hagan trampas, se automatizan todos los pasos a seguir en el juego y se hace mucho más fácil de empezar a jugar.

Componentes:

- Tablero del juego
- Elementos físicos:
    - Cartas de evento (amenazas, objeto y compañero)
    - Cartas de elementos básicos
    - Fichas jugadores
- Cartas virtuales
    - Cartas de acción
    - Objetos combinables

Tablero:

Representa el mapa alrededor del centro de investigación que da el aviso y los búnkeres colindantes. Está dividido en seis zonas, siendo una de ellas el centro de investigación y el resto las de los cinco búnkeres. Cada zona tiene un material primario abundante y están subdivididas en seis áreas con diferentes características útiles para los jugadores.

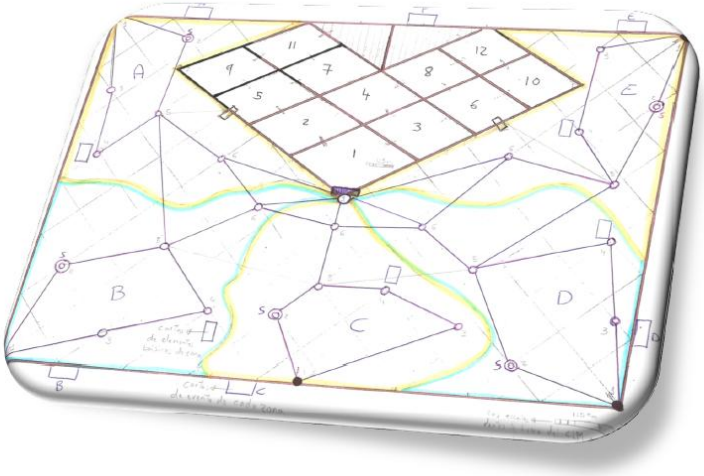


Ilustración 1. Tablero del juego

1.3 Fases del proyecto

En este apartado se definirán con detalle y en orden de ejecución, cada una de las etapas que se han seguido durante el desarrollo del proyecto que nos ocupa. Con ellas se pretende optimizar el proceso y el producto software. Las fases del proyecto [11] coinciden con las que normalmente se utilizan en la mayoría de proyectos software:

1. **Análisis:** La tarea principal de esta fase es la captura de requisitos. Estos deben delimitar exactamente cómo ha de funcionar la aplicación. Se realizarán los controles que sean necesarios para asegurar la coherencia del producto y valorar la dificultad del proyecto. Si fuera necesario, se realizarán nuevas reuniones con el cliente a fin de tener una base más solida sobre la que comenzar el diseño y el desarrollo.

2. **Diseño:** La siguiente tarea a llevar a cabo, tras la fase de análisis, consiste en realizar el diseño completo de la aplicación. Esto implica el diseño de las bases de datos que almacenarán la información, diseño de navegación y de la interfaz de usuario y diseño de las clases que se utilizarán (Dado que gran parte de la aplicación se va a desarrollar en Java).

3. **Implementación:** Durante esta fase se codificarán los diseños definidos durante la fase anterior. Además, se realizarán parte de las pruebas que más adelante habrá que documentar. Como se explica a continuación en el modelo de ciclo de vida escogido, no supondrá un gran inconveniente realizar modificaciones en esta fase, ni en las anteriores.

4. **Pruebas:** Aunque durante la fase de implementación se realizarán muchas de las pruebas que se van a llevar a cabo, en esta se tratará de completar ese proceso para asegurar que todas las funciones, módulos y partes de la aplicación realizan exactamente su función como el cliente desea, probando todos los casos y documentando todos los resultados obtenidos. También se realizarán partidas de prueba para asegurar que la aplicación responde como se desea y que los valores establecidos son coherentes y no afectan negativamente al juego.

5. **Documentación:** A parte de toda la documentación de requisitos, de diseño y de código que se ha ido realizando durante las fases anteriores, se deberá llevar a cabo una documentación general y más completa, en la que se recogerán todos los aspectos del proyecto y se relacionarán todos los puntos importantes de las fases, así como los resultados de las mismas y las conclusiones obtenidas.

6. **Mantenimiento:** Normalmente, esta es una de las fases más largas de los proyectos software. Se realizan todo tipo de mejoras, actualizaciones y modificación en la aplicación. En nuestro caso, no es seguro que nos encarguemos nosotros de esta fase y por eso es importante dejar el proyecto bien documentado y estructurado.

Una vez definidas las fases del proyecto y el orden que van a seguir en el proceso software, y habiendo establecido los criterios de transición entre cada una de ellas (entradas y salidas que se a denominarán con el nombre de productos) se procederá a justificar el ciclo de vida [12] utilizado en este proyecto.

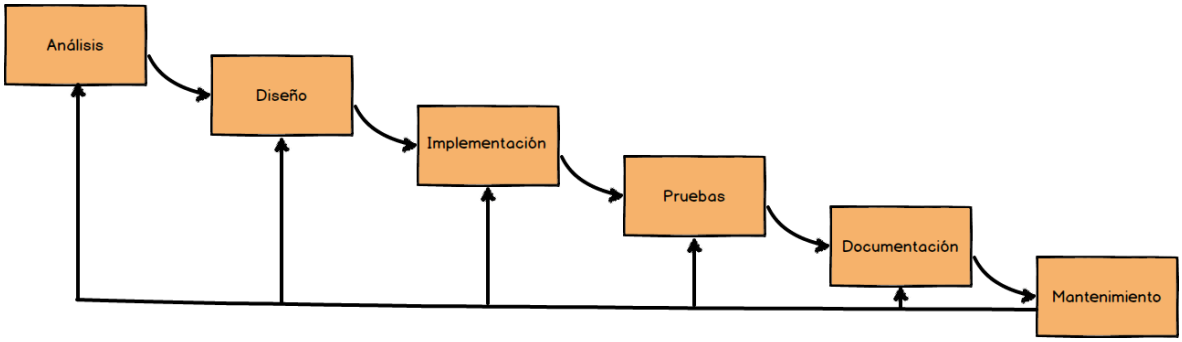


Ilustración 2. Modelo de ciclo de vida

Se ha escogido una variante bastante común del modelo de ciclo de vida en cascada, añadiéndole una mejora iterativa que permite ir generando los productos de las diferentes etapas e irlos refinando y mejorando. Con ello se evitan algunas de las limitaciones que se producen al utilizar el modelo de cascada clásico. El principal problema que ha obligado a utilizar este modelo es que los requisitos al principio del proyecto no estaban muy bien definidos y mostraban ambigüedades. Para evitar un enorme parón al principio del proyecto, se decidió empezar a realizar las siguientes etapas a partir de los requisitos que estaban más claros, teniendo la seguridad de poder volver a las etapas anteriores cuando fuera necesario. Una vez fijado los requisitos, se repetirán las fases de diseño, desarrollo y pruebas hasta cumplir los resultados esperados.

1.4 Estructura del documento

Este documento recoge todas las etapas descritas en el apartado anterior. Para cada una de ellas se ha reservado un capítulo en el que serán descritos todos sus puntos clave, características y decisiones importantes que hayan afectado en la generación de sus productos.

En el capítulo *2.Estado del arte,* se va a exponer toda la información que se ha considerado de interés acerca de Android, de las herramientas que se han utilizado para mejorar el proyecto y de otros proyectos similares cuyo objetivo principal es el de combinar los juegos de mesa con las nuevas tecnologías móviles.

En el capítulo *3.Análisis de requisitos,* se describirán los tipos de requisitos a tener en cuenta para el proyecto, los resultados del proceso de captura y la división en módulos diferenciados para mejorar el producto final.

En el capítulo *4.Diseño de la solución,* se documentarán todas las decisiones de diseño que han sido tomadas, como el patrón de diseño, estructura y tipos de los datos, acompañadas de los diagramas que describen mucho mejor como funcionará la aplicación final.

En el capítulo *5.Implementación* se explicarán todos los conceptos de Android y de las demás herramientas utilizadas que faciliten el entendimiento de cómo ha sido codificada la aplicación.

En el capítulo *6.Pruebas y resultados,* se presentarán las pruebas que se han realizado sobre la aplicación y los resultados obtenidos de estas. Esto incluirá varios tipos de pruebas para asegurar que la aplicación y sus componentes responden correctamente y como el cliente desea.

Por último, el capítulo *7.Conclusiones y trabajo futuro*, incluirá las conclusiones obtenidas de la realización completa de la aplicación y se comentarán posibles trabajos futuros que puedan aportar mejoras al proyecto software.

Anexos información para una mejor comprensión del documento que, por su extensión, se han decidido no incluir como contenido de la memoria.

2 Estado del arte

En este capítulo, como trabajo previo de investigación para mejorar el producto final del proyecto, se ha llevado a cabo un breve estudio acerca de Android y del mercado de los dispositivos móviles, donde se detallan sus características con el objetivo de esclarecer porqué se ha decidido enfocar la aplicación solo para este sistema operativo. Se mostrará información sobre cómo funciona Android, sus inicios, que avances le han hecho llegar tan lejos, informes estadísticos de los usuarios de Android frente a sus competidoras y se tratarán además temas como las versiones, características y utilidades que se usarán durante el proyecto.

Parte de este trabajo de investigación es la búsqueda de aplicaciones similares en las que poder basarse para mejorar los resultados. No solo en las tecnologías móviles si no también otros intentos de combinar juegos de mesa con las nuevas tecnologías.

Además, para este apartado se describirán algunas de las herramientas más importantes y que más útiles han resultado para el desarrollo de la aplicación. Algunas de ellas ya se conocían de trabajos previos y otras se han tenido que aprender a utilizar. Este aprendizaje formará parte del trabajo previo de investigación realizado.

2.1 Android

Android es la plataforma móvil más importante del mundo, utilizada por cientos de millones de dispositivos móviles en más de 190 países, proporciona una fenomenal base para la creación de aplicaciones y juegos para sus usuarios, así como un mercado abierto para su distribución casi instantánea.

Android es un sistema operativo multiplataforma, libre y gratuito, basado en el Kernel de Linux y diseñado principalmente para dispositivos móviles con pantalla táctil. El SDK (Software Development Kit) de Android proporciona todas las herramientas necesarias para poder desarrollar aplicaciones utilizando Java. Fue desarrollado por una startup que más tarde adquiriría Google. Al tratarse de un software libre, característica muy atractiva para muchos, los usuarios y los fabricantes pueden crear sus propias extensiones y mejorar enormemente la plataforma.

Como se puede observar en la *Ilustración 3* los componentes principales [2] del sistema operativo Android son los siguientes:

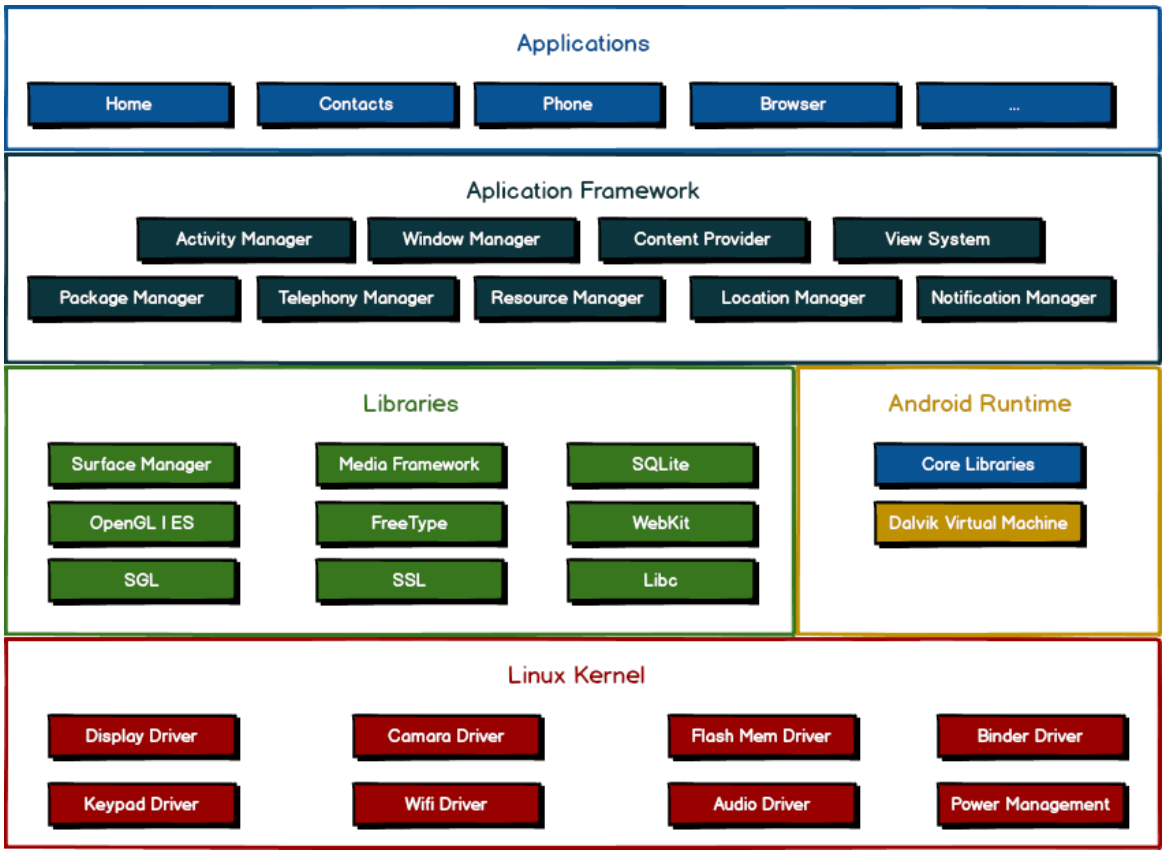


Ilustración 3.Componentes del SO Android

- **Núcleo de Linux:** Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también se encarga de comunicar el hardware y el resto de capas de la arquitectura.

- **Runtime de Android:** Es la capa que está compuesta por las librerías y la maquina virtual que utiliza Android. Esta se conoce por el nombre de Dalvik, una variante de la maquina virtual de Java diseñada para optimizar la ejecución concurrente de varias aplicaciones.

- **Librerías:** Android incluye un conjunto de bibliotecas en C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android. Algunas importantes son bibliotecas de medios, 3D y SQLite.

- **Marco de trabajo de aplicaciones:** los desarrolladores pueden utilizar los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar su reutilización y cualquier usuario puede publicar nuevas para ser utilizadas por el resto de usuarios.

- **Aplicaciones:** Cualquier aplicación instalada en el dispositivo.

Desde sus comienzos, la evolución de Android ha sido constante y esto ha dado lugar a numerosas versiones [3] como puede observar en la siguiente tabla.

Versión	Fecha de publicación	API Level	Nombre
Beta	Noviembre 2007		
1.0	Septiembre 2008	1	Astro
1.1	Febrero 2009	2	Bender
1.5	Abril 2009	3	Cupcake
1.6	Septiembre 2009	4	Donut
2.0 – 2.1	Octubre 2009	5, 6 y 7	Eclair
2.2	Mayo 2010	8	Froyo
2.3	Diciembre 2010	9 y 10	Gingerbread
3.0 – 3.2	Febrero 2011	11, 12 y 13	Honeycomb
4.0	Octubre 2011	14 y 15	Ice Cream Sandwich
4.1	Julio 2012	16	Jelly Bean
4.2	Noviembre 2012	17	Jelly Bean
4.3	Julio 2013	18	Jelly Bean
4.4	Octubre 2013	19	Kitkat

Tabla 1. Versiones de Android

La aplicación a desarrollar para este proyecto está enfocada a dispositivos con una versión de Android 4.0 o superior. Por ello algunas de las funcionalidades que se incluirán en el proyecto pueden no funcionar correctamente en versiones anteriores. Para el desarrollo y las pruebas se han utilizado un móvil *Xperia U* con la 4.0, un móvil *Samsung Galaxy Ace 2* con la versión 4.1 y la tableta *Samsung Galaxy Note 10.1* con la 4.4.

2.2 Trabajo previo de investigación

Los primeros pasos del estudio de investigación realizado han estado enfocados al sistema operativo elegido para la realización de este proyecto. Como se puede observar en la siguiente imagen, desde que Android comenzase su andadura, en tan solo tres años alcanzó el primer puesto en la venta de teléfonos inteligentes y a estas alturas ya supera con creces a todos sus competidores con más del 80% de la cuota de mercado y en algunos trimestres posteriores, que no se muestran en la gráfica, llegando incluso al 90%. Con estos datos queda bastante claro porque se ha elegido Android y no sus competidores IOS o WindowsPhone para el desarrollo de la aplicación.

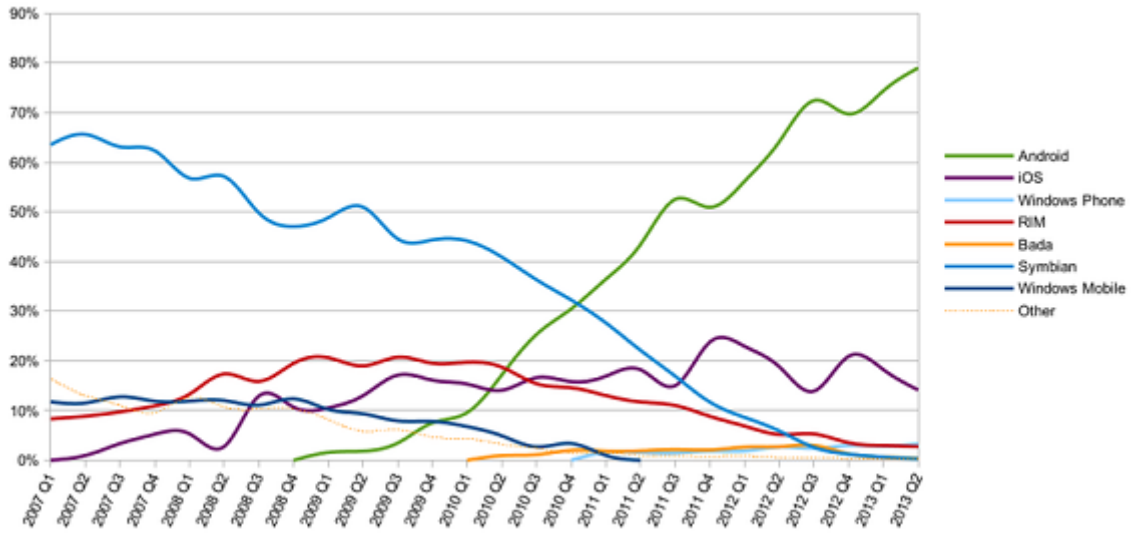


Tabla 2. Porcentajes de ventas de Smartphones

Para esclarecer como se tomó la decisión sobre las versiones que abarcaría la aplicación nos remitimos al siguiente gráfico, en el que pueden observar el porcentaje de dispositivos móviles que utilizan cada versión. Este proyecto, como ya se ha comentado anteriormente, está enfocado para la versión 4.0 o superiores. Con esta elección de versiones abarcamos aproximadamente un 85% de los dispositivos Android que se venden en la actualidad [13].

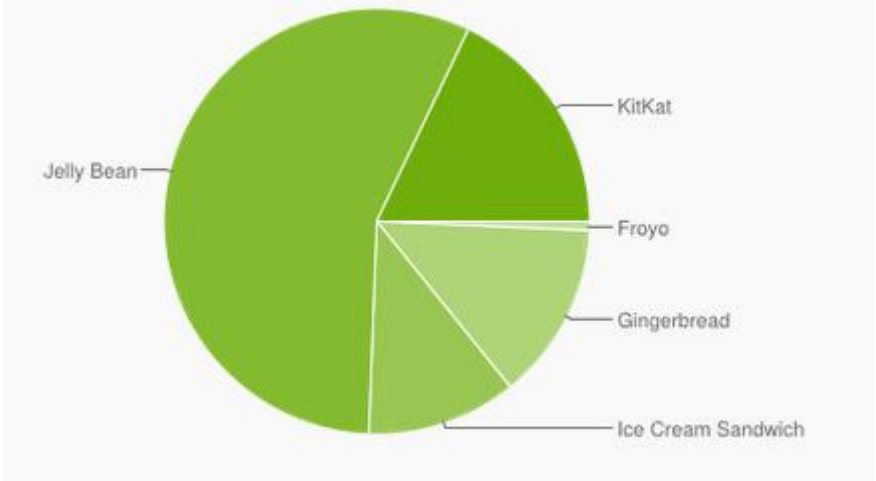


Ilustración 4. Porcentajes de utilización de versiones Android

Otra parte muy importante del estudio realizado, ha sido la búsqueda de aplicaciones similares que tomar como punto de partida y sobre las que trabajar para mejorar el resultado final del proyecto. Algunas de ellas no parecen estar relacionadas en absoluto con lo que se quiere conseguir, pero algunos de sus aspectos de jugabilidad han resultado útiles para el diseño del juego. Algunos ejemplos de juegos que resultaron provechosos fueron:

- **Atmosfear:** Uno de los primeros juegos de mesa [14] en combinarse con las nuevas tecnologías, en este caso las cintas de video VHS. Al tratarse de un juego de mesa relativamente antiguo, evidentemente no es una aplicación ni nada por el estilo. Lo que resulta atractivo para nuestro proyecto es como se encarga el juego, a través del video, de marcar los objetivos a los jugadores dándole una pizca de realidad que aumenta el ambiente de juego en cada partida.

- **Apalabrados, DrawSomething, entre otros:** Estos juegos, bastante más recientes, han conseguido trasladar los juegos de mesa completamente a los dispositivos móviles. Algunos aspectos interesantes de estas aplicaciones son la gestión de turnos y la UI mediante los que se han conseguido juegos muy dinámicos y entretenidos. De todas formas, siguen sin cumplir exactamente lo que se busca con este proyecto.

- **Descent Quest Vault:** Como en el caso anterior, consiste en un juego de mesa para jugar únicamente desde el teléfono, pero lo que de verdad resulta llamativo de este juego es que los usuarios pueden crear y descargar nuevas aventuras y así conseguir un mayor tiempo de juego. Para nuestro proyecto se quiere evitar partidas monótonas y siempre parecidas y la aplicación deberá encargarse gestionar algunos elementos para conseguir que cada partida sea diferente.

- **App-Player:** Lo maravilloso de este juego [15] es que dispone de un tablero muy bien diseñado que se adapta a numerosas aplicaciones. Con una especie de juegos reunidos para los que simplemente habrá que descargarse una nueva aplicación y jugar con el mismo tablero de siempre.

2.3 Herramientas y tecnologías utilizadas

A continuación se describirán algunas de las herramientas utilizadas durante el desarrollo de este proyecto software.

- **Eclipse SDK:** Programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar aplicaciones para un determinado paquete de software, estructura de software, plataforma o similar. Incluyen herramientas de debugger, ejemplos, documentación, y muchas veces un entorno de programación integrado (IDE).

- **Assembla:** Empresa que provee herramientas de colaboración y de seguimiento de errores y tareas basadas en la nube, para organizar y administrar proyectos de código abierto y comerciales para el desarrollo de software. Muy importante para poder trabajar conjuntamente con el desarrollador encargado de la parte del controlador.

- **Dropbox y OneDrive:** Servicios de alojamiento de archivos multiplataforma en la nube. El servicio permite a los usuarios almacenar y sincronizar archivos en línea y entre ordenadores, para así compartir archivos y carpetas con otros. Se han utilizado dos porque OneDrive permite la edición online de documentos Word.

- **Visual Paradigm:** Herramienta de diseño de software que soporta los estándares de modelado como UML, entre otros. Facilita el desarrollo de software y sistemas.

- **Apache 2:** Servidor web HTTP de código abierto para plataformas como Linux, Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Será utilizado para crear el servidor local de pruebas de la aplicación.

- **MySQL:** Sistema de administración de bases de datos [6] relacional, multihilo y multiusuario. Utiliza múltiples tablas para almacenar y organizar la información y destaca por su gran adaptación a diferentes entornos de desarrollo, permitiendo su interactuación con los lenguajes de programación más utilizados como PHP, Perl y Java y su integración en distintos sistemas operativos.

- **PHP:** Un lenguaje de programación [5] de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico

- **PhpMyAdmin:** Herramienta de software libre escrito en PHP, con la intención de mejorar la administración de MySQL a través de la Web. Compatible con una amplia gama de operaciones en MySQL. Operaciones de uso frecuente (gestión de bases de datos, tablas, columnas, relaciones, índices, usuarios, permisos, etc.) se puede realizar mediante la interfaz de usuario.

- **SQLite:** Es un motor de bases de datos [16] muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y por supuesto ser de código libre.

3 Análisis de requisitos

Un requisito es una circunstancia o condición necesaria para que se produzca algo, en este caso concreto se trata de una necesidad documentada del comportamiento o funcionalidad esperado de un producto. Se distinguirán dos tipos [2] principales de requisitos:

- **Requisitos funcionales:** Características del sistema que expresan una capacidad de acción del mismo. Una funcionalidad.

- **Requisitos no funcionales:** Características del sistema, del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que señala una restricción del mismo.

3.1 Requisitos funcionales

Se dividirán los requisitos funcionales en subcategorías para facilitar su entendimiento y su posterior desarrollo. Con ellos además se conseguirá una mejor comunicación del equipo y una mejor conformidad con los estándares Android. Nótese que los requisitos que se detallan a continuación están orientados al alojamiento de la información y a la interfaz del usuario.

3.1.1 Gestión de usuarios

- **RF01: Registrar usuarios.** Para darse de alta como nuevo usuario en el sistema será necesario introducir toda la información personal requerida (email, apodo, contraseña, etc.) desde una única pantalla de registro. La aplicación realizara las validaciones y chequeos pertinentes con el servidor y éste generará un identificador único para el usuario. Al finalizar el proceso, el usuario habrá sido registrado y autenticado en el servidor y por tanto su sesión estará activa.

- **RF02: Autenticar usuarios (Log-in).** Para iniciar sesión el usuario deberá introducir su nombre y contraseña desde la pantalla de inicio de sesión. La aplicación comprobará los datos con el servidor y según la respuesta de éste autenticará al usuario o le mostrará un mensaje de error.

- **RF03: Cerrar sesión de usuario.** Un usuario podrá abandonar su sesión en cualquier momento, siempre y cuando no se encuentre jugando una partida (deberá guardar la partida y salir de ella antes de hacerlo). Todos los datos seguirán estando disponibles en el servidor para cuando vuelva a autenticarse. Al cerrar la aplicación no se abandona la sesión y no se pierde ninguna información.

- **RF04: Mostrar información de usuario.** Cualquier usuario con sesión activa podrá acceder desde cualquier parte de la aplicación a su información de jugador. Esta consiste principalmente en su información personal y algunas estadísticas que se irán recogiendo durante las partidas.

- **RF05: Modificar información de usuario.** Desde la cuenta de usuario donde se muestra su información también se ha de permitir modificar algunos campos como pueden ser la contraseña, el apodo, el correo, etc.

- **RF06: Dar de baja Usuarios.** Siempre que se desee, un usuario puede eliminar su cuenta por las circunstancias que sean. Se marcará su cuenta como inactiva, aunque no se borrará toda su información. Con ello se pretende que afecte en la menor medida posible a las partidas que tenía abiertas con otros jugadores y no se pierda todo su progreso por si quisiera volver a jugar.

3.1.2 Gestión de partidas

- **RF07: Creación de nuevas partidas.** Cualquier usuario logueado podrá crear las partidas que desee desde el menú de la aplicación. Para ello deberá introducir un nombre de partida y si lo desea una contraseña para evitar que entren jugadores desconocidos. Una vez se crea la partida, se redirigirá al creador a una pantalla de espera para el resto de jugadores. Si el jugador que crea la partida, abandona esta pantalla antes de que empiece la partida, esta no se guardará definitivamente en el servidor.

- **RF08: Unirse a partidas creadas.** Cualquier jugador logueado deberá poder unirse a las partidas disponibles. Cuando se crea una partida se da por supuesto que el creador esta unido a la partida.

- **RF09: Permisos en la creación de partidas.** El usuario que crea la partida puede restringir los jugadores que se unen a ella mediante una contraseña pero además también de poder eliminar a jugadores unidos en la pantalla de espera. Un ejemplo puede ser alguien que se haya unido adivinando la contraseña y el administrador de la partida no quiere que juegue.

- **RF10: Proteger las partidas.** En línea con los RF08 y RF09 el creador y administrador de las partidas puede protegerlas mediante una contraseña, que deberá ser especificada antes de crear la partida. Todos los jugadores que deseen unirse a la partida deberán introducirla correctamente.

- **RF11: Guardar partidas.** Como en cualquier juego de mesa, las partidas deberán poder dejarse a medias. La aplicación deberá ser capaz de almacenar todos los datos necesarios de la partida que permitan a los jugadores retomar la partida tal cual la dejaron.

- **RF12: Cargar partidas guardadas.** En línea con el RF11 las partidas guardadas se deben poder recargar para continuar jugando desde el mismo punto en el que se dejo al guardar.

- **RF13: Abandonar partidas.** En cualquier momento un jugador deberá poder eliminarse de una partida. La partida continuara activa mientras siga manteniendo el número mínimo de jugadores. Eliminar parte de la información no debería afectar a los datos del resto de jugadores. Si la partida no alcanza el mínimo número de jugadores se notificará y posteriormente se eliminara.

3.1.3 Gestión de elementos de la partida

- **RF14: Gestión de eventos.** La aplicación deberá ser capaz de gestionar todos los tipos de eventos que puedan surgir a lo largo de la partida. Tendrá que dar acceso a la información de estos, que estará alojada en la base de datos del móvil, diferenciar los distintos tipos y realizar las acciones pertinentes en función del tipo de evento.

- **RF15: Gestión de objetos.** El primer tipo de eventos serán los objetos. La gran mayoría de ellos estarán asociados a una carta física de nuestro juego de mesa, aunque existirán algunos que se construirán a partir de combinaciones y solo estarán alojados en la base de datos. La aplicación tendrá que poder acceder a esta información cuando sea necesario y ser capaz de identificar las combinaciones de objetos que se pueden llevar a cabo.

- **RF16: Gestión de recetas.** La base de datos de la aplicación debe ser capaz de alojar todas las combinaciones de componentes que darán lugar a otros objetos y disponer de los métodos necesarios para realizarlas durante las partidas.

- **RF16: Gestión de compañeros.** Al principio de cada partida la aplicación tendrá que asignar a cada jugador un compañero aleatorio en función de sus características. Durante las partidas podrán realizarse cambios de compañeros y estos podrán ser eliminados por los rivales. La aplicación gestionará toda esta información desde la base de datos y a través del servidor cuando tenga que almacenar los datos de las partidas.

- **RF17: Gestión de amenazas.** Durante las partidas se generarán automáticamente, y de forma aleatoria, amenazas para todos los jugadores. El sistema deberá ser capaz de realizar búsquedas aleatorias en la base de datos, generar las amenazas y si fuese necesario, avisar a los jugadores sobre ellas.

- **RF18: Información estadística.** En cada partida se irá guardando información acerca de los combates, los objetos y las decisiones que toma un jugador en el servidor. Esta información deberá ser accesible desde cualquier dispositivo en el que el jugador inicie sesión.

3.1.4 Gestión de la aplicación

- **RF19: Cambiar idioma.** La aplicación deberá ser multi lenguaje. Desde el menú de ajustes de cualquier Smartphone se puede cambiar la configuración del idioma al que se desee. Android facilita el cambio de idioma del juego traduciendo todas las cadenas de caracteres al idioma seleccionado en los ajustes. Se requiere por lo menos español e inglés siendo ampliable a francés e italiano.

- **RF20: Preferencias.** Desde cualquier pantalla de la aplicación se podrá acceder al menú de ajustes y preferencias del juego. Será necesarios que las elecciones que realice el jugador cada vez que modifique alguna se guarden permanentemente en la aplicación, persistiendo esta información aunque se cierre por completo la aplicación. En caso de desinstalación estos datos se perderán y si se vuelve a instalar se pondrán los valores por defecto.

3.2 Requisitos no funcionales

- **RNF01: Mantenibilidad.** El diseño de la aplicación seguirá el modelo vista controlador. Con esto se facilitarán los análisis y las modificaciones que puedan realizarse en el futuro sobre el proyecto. Al estar dividido en módulos diferenciados y siempre separando la apariencia de la funcionalidad, las pruebas serán más naturales y claras. Todo el código será debidamente comentado y se procurará utilizar los estándares de programación de Android.

- **RNF02: Portabilidad.** El software será utilizable por cualquier dispositivo táctil Android, dando igual su tamaño o prestaciones, siempre y cuando opere con una versión superior a la 4.0 Ice Cream Sandwich. Será completamente necesario disponer de una conexión a internet, ya sea por Wifi o por tarifa de datos.

- **RNF03: Usabilidad.** Se procurará que la interfaz sea simple y atractiva, no queremos que el usuario interaccione constantemente con el sistema, si no que sirva de apoyo al juego de mesa y que cualquier operación a realizar no implique demasiado tiempo. Será conveniente que los usuarios tengan experiencia con otras aplicaciones móviles. El modo de interactuar con los menús será sencillo, pero a la hora de jugar partidas, la interfaz variará bastante y será menos intuitiva, por lo menos al principio.

- **RNF04: Fiabilidad.** Se asegura la integridad de los datos de la aplicación ante cualquier problema que pueda surgir: fallo de conexión, cierre imprevisto o dejar la aplicación en segundo plano. Todos los datos facilitados por el usuario serán protegidos mediante cifrado y control de accesos y no se utilizarán ni serán facilitados a ninguna aplicación o entidad externa.

- **RNF05: Rendimiento.** Al tratarse de un primer prototipo de la aplicación el servidor será de pruebas y no será capaz de responder a muchas peticiones concurrentemente. Por supuesto, esto también dependerá de la conexión a internet de la que se disponga. Se esperan tiempos de respuesta no superiores a 2 segundos en las peticiones al servidor y menores aún, en las consultas a la base de datos. Los accesos a la base de datos y los cálculos que se realicen en la aplicación son rápidos y no implican demasiado esfuerzo por parte del dispositivo y por tanto el rendimiento en estos casos será óptimo, dependiendo siempre de las prestaciones.

4 Diseño de la solución

El diseño es el proceso de definición de la arquitectura, componentes, módulos, interfaces, procedimientos de prueba y datos de un sistema software para satisfacer unos requisitos ya especificados. Para la realización de un buen diseño es conveniente dividir el problema en módulos, de forma que cada uno de éstos sean tan independientes (mínimo acoplamiento) y específicos (máxima cohesión) como sea posible.

En Android se utiliza el patrón de diseño modelo vista controlador [17] (MVC) que permite separar los datos de una aplicación, la interfaz de usuario y la lógica en tres componentes independientes. Esto facilita enormemente el trabajo cuando se desarrollan proyectos grandes y en equipo.

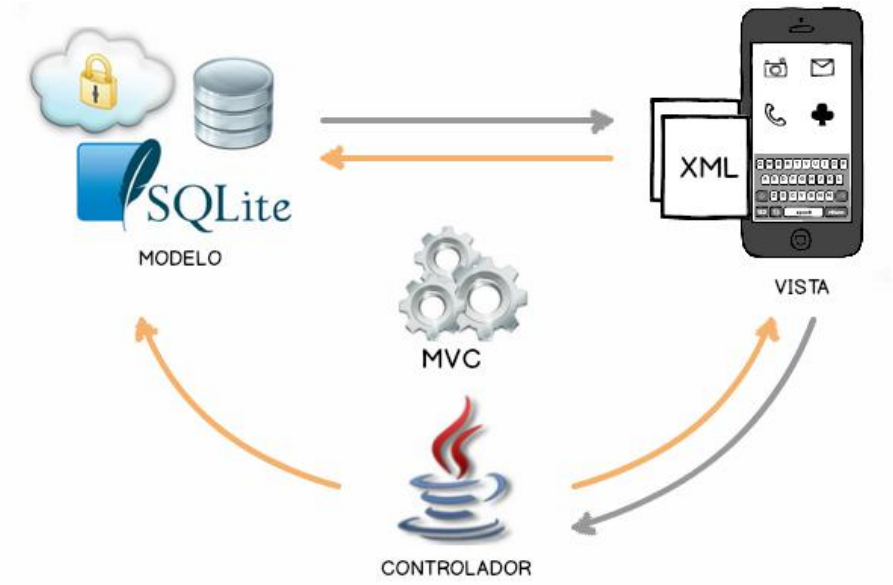


Ilustración 5. Modelo vista controlador

A continuación se explicarán cada una de las secciones antes definidas, aunque este documento se centrará especialmente en el modelo y en la vista.

- **Modelo:** Se refiere a la forma de almacenar toda la información con la que operará la aplicación. Toda la información fija y que no depende de los usuarios será alojada en la base de datos SQLite de los dispositivos móviles, mientras que el resto estará disponible a la aplicación a partir de un servidor web LAMP [4].
- **Vista:** Consiste básicamente en la interfaz con la que interactúan los usuarios y a través de la cual realizan todas la acciones disponibles. En Android utilizaremos XML para construir y dar formato al esqueleto de nuestra aplicación.
- **Controlador:** Para terminar, el controlador constituye todas las clases y funciones Java que permitirán a la aplicación interactuar entre el modelo y la vista, realizando las consultas y modificaciones de la información que sean necesarias y gestionando sus efectos sobre la interfaz de usuario (UI).

Para empezar, se divide el diseño completo en diseño en interfaz, las estructuras de datos internas y las estructuras de datos externas como se puede observar a continuación.

4.1 Diseño de la interfaz

Para el diseño de la interfaz de usuario se ha decidido diferenciar dos partes principales. La primera, que siempre se mostrará al iniciar la aplicación, estará formada por todos los menús y las opciones de la aplicación. Una vez que el usuario cree o retome una partida se pasará a la parte del juego. Ambas partes estarán claramente diferenciadas tanto en el diseño como en el desarrollo ya que apenas comparten funcionalidades. A continuación se muestra el mapa de navegación definitivo de la parte de los menús de la aplicación.

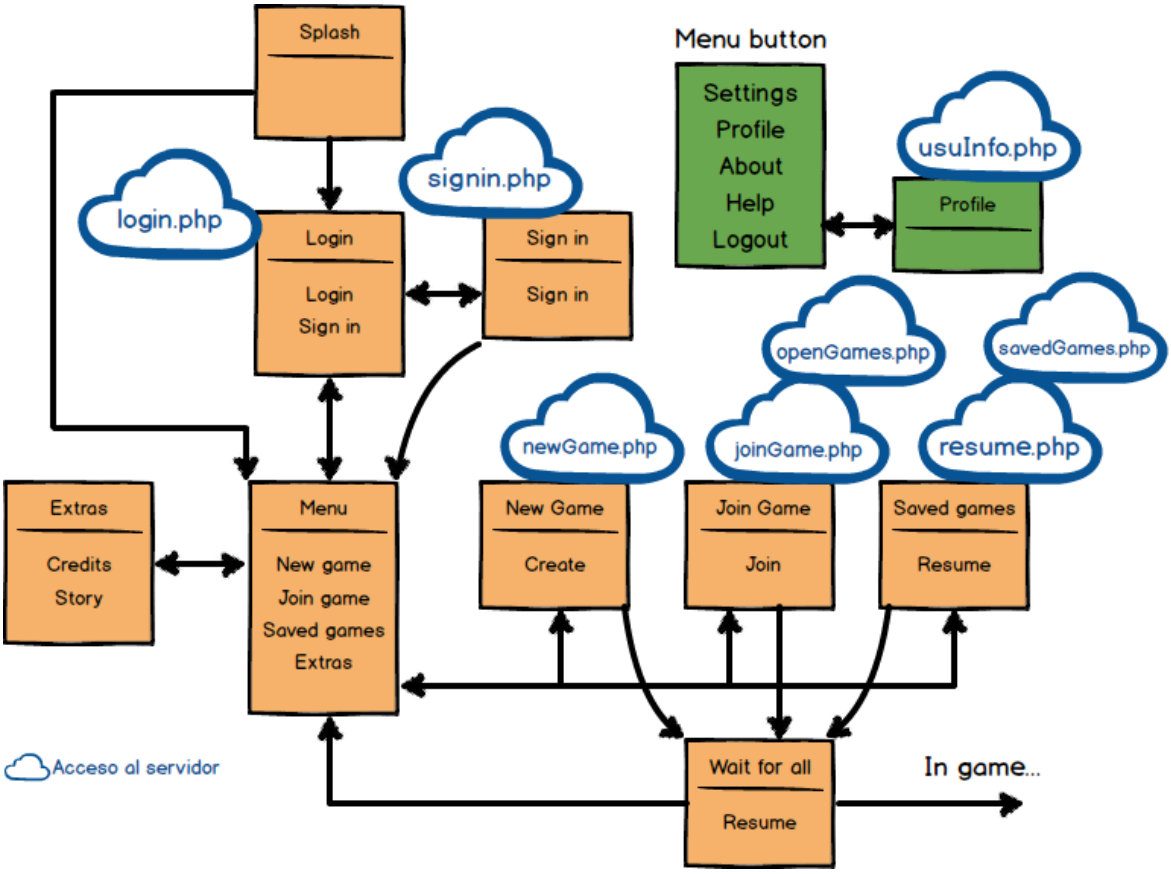


Ilustración 6. Diseño navegación de los menús

Los recuadros en color beis simbolizan las pantallas reales que formarán la aplicación y las que están en color verde serán las pantallas del menú de opciones dentro del juego. Se podrá acceder desde cualquier pantalla a este menú de opciones. La navegación dentro de la aplicación queda delimitada por las flechas que se muestran en la imagen.

Cada pantalla que así lo requiera, accederá a la información que necesite del servidor a partir de scripts PHP. Como se puede ver en la imagen, simbolizados por nubes con los respectivos nombres de los scripts.

La transición de esta primera parte de la aplicación a la parte del juego queda indicada por la fecha *In game* que sale de la pantalla de espera.

Una vez se empieza o retoma una partida se mostrará la pantalla principal de la parte del juego. La navegación es algo diferente a la anterior, se puede acceder a todas las pantallas desde todas las pantallas y la gran mayoría de las interacciones con el servidor también se deben poder ejecutar desde cualquier pantalla. Esto implica que las funciones deben ser totalmente independientes para evitar repetir código más adelante. En la siguiente ilustración se puede observar cómo funcionará esta parte.

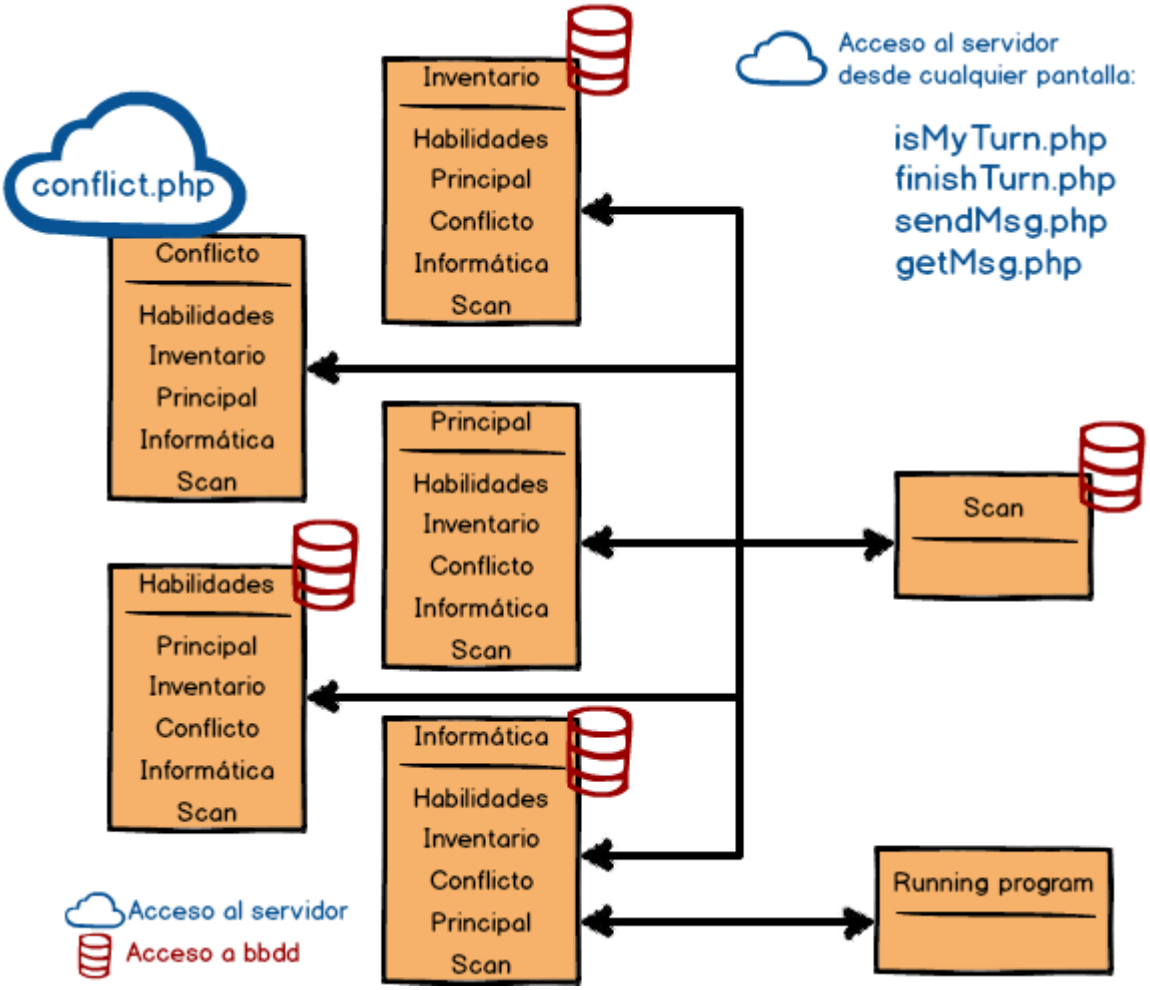


Ilustración 7. Diseño navegación del juego

Al igual que en la parte de los menús, los accesos a información del servidor quedan representados por las nubes, pero además, esta parte del juego cuenta con accesos a información de la base de datos interna del dispositivo. Esta información será básicamente la información fija de la aplicación como objetos, amenazas y compañeros.

Para el patrón escogido ambos diseños formarán parte de la vista, que junto con algunas funciones del controlador, constituirán el esqueleto básico de navegación de la aplicación. La distribución y el estilo de cada uno de los elementos de cada pantalla se implementarán en los ficheros de diseño XML.

4.2 Diseño de la estructura de datos

A continuación mostrará cómo se han diseñado las estructuras de datos destinadas a alojar la información de nuestra aplicación. Se separarán claramente las estructuras de datos internas del dispositivo de las del servidor, ya que no mantienen ninguna relación entre ellas y los accesos son totalmente diferentes.

4.2.1 Diseño de la base de datos interna

Toda la información almacenada en la base de datos del dispositivo será fija, es decir, no sufrirá ningún cambio en ningún caso de uso de la aplicación. Serán los desarrolladores los que modifiquen la información mediante actualizaciones de la aplicación. Android utiliza el motor de bases de datos transaccional SQLite que no precisa de un servido separado. En la siguiente imagen se puede observar el diseño final de la base de datos.

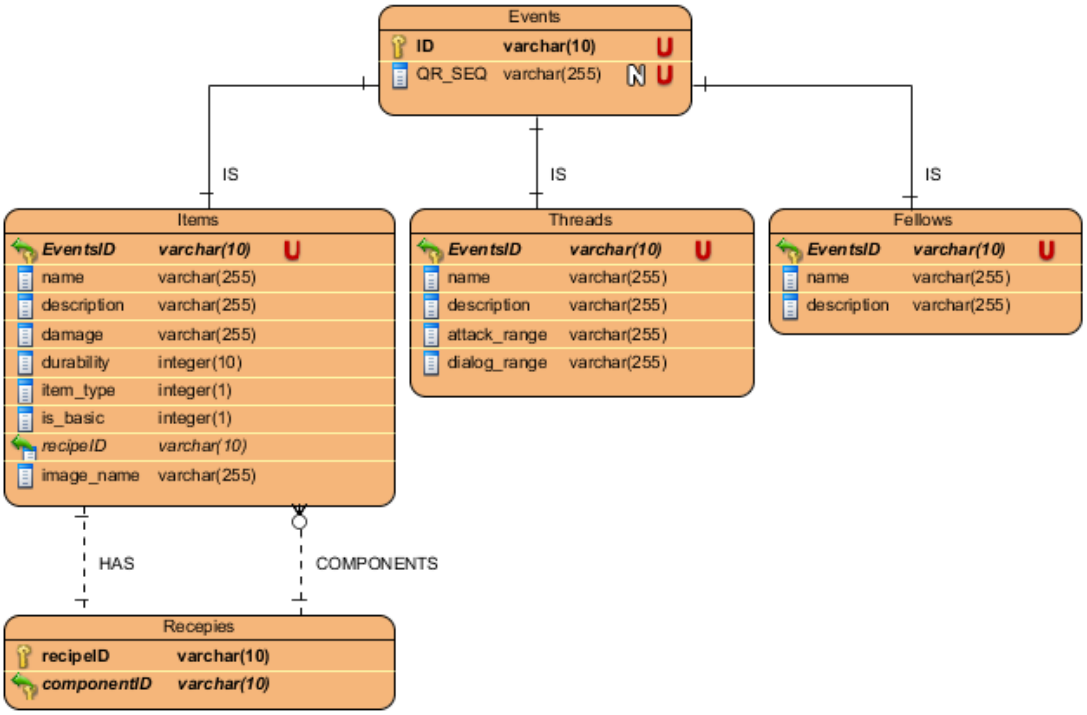


Ilustración 8. Diseño relacional de la bbdd

Los eventos son el elemento principal de la base de datos interna. Objetos, amenazas y compañeros son tipos de eventos que añaden nuevas características al mismo. Cada evento consta de un identificador, y cuando se corresponden con una carta física del juego, un código QR que se utilizará para indicar a la aplicación que evento se está procesando. Cada subtipo tiene su propio identificador, que referencia al del evento, y características como nombre, descripción, tipo, entre otras. Como ya se ha dejado intuir, no todos los eventos tienen asignada un elemento físico del juego. Es el caso de los objetos combinables que solo podrán conseguirse completando su receta. Las recetas se referencian desde el objeto que crean y contienen todos los componentes (otros objetos) que se necesitan para crearlo.

4.2.2 Diseño del servidor

En el servidor se va a almacenar toda la información susceptible a modificaciones por las acciones que realizan los jugadores. El servidor va a estar alojado en un ordenador con Ubuntu, las bases de datos estarán en MySQL y los accesos se gestionarán mediante métodos asíncronos y scripts PHP. La estructura de la base de datos del servidor es la siguiente:

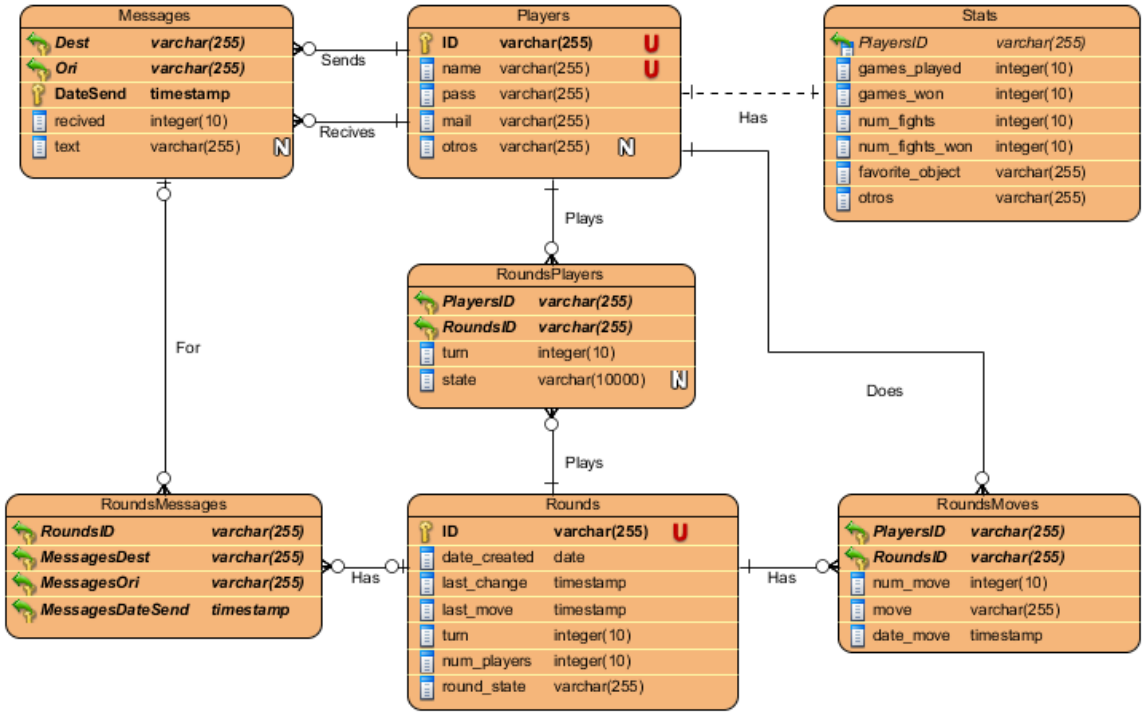


Ilustración 9. Diseño relacional de la bbdd del servidor

Cada jugador registrado con su Nick, contraseña y datos personales tendrá asignado un identificador único dentro de la base de datos. Asignadas a cada jugador, el servidor administrará las estadísticas de las partidas que juegue. Para cada partida que se crea se guardara un registro en la tabla *Rounds* con un identificador, fecha de creación, últimos cambios, turno actual y el estado de la partida (creándose, lista, jugándose, terminada). Cada jugador que participe en una partida tendrá otro registro en la tabla *RoundsPlayers* con su turno y estados dentro de la partida. Esta información se actualiza durante el juego y se utilizará más adelante para cargar las partidas. Además para cada partida se almacenan todos los movimientos realizados, para asegurar que la gestión de turno se realiza de forma correcta. Por último, y para permitir el intercambio de información entre los dispositivos de los jugadores, se implementa un mecanismo de mensajes a través de los cuales se gestionarán los conflictos, las amenazas y los diálogos del juego. Los mensajes podrán dirigirse tanto a los integrantes de una partida como a un jugador en concreto.

Para este proyecto (sin contar el trabajo posterior al TFG) las funcionalidades del servidor no van a estar integradas en la aplicación. Se implementarán y probaran de forma independiente y se utilizarán datos locales para asegurar que el resto de la aplicación funciona correctamente.

5 Implementación

Android proporciona un rico framework de aplicaciones que permiten construir todo tipo de herramientas y juegos para dispositivos móviles en lenguaje de programación Java. El SDK de Android compila el código, junto con toda la información de los ficheros de recursos, para crear un *APK* (Android Package). Un fichero *APK* contiene toda la información necesaria para instalar la aplicación en un terminal que utilice Android.

Una vez instaladas en un dispositivo, cada aplicación es independiente al resto. El SO de Android es un sistema multiusuario de Linux, en el que cada aplicación es un usuario diferente. El sistema asigna a cada uno su propio ID y administra los archivos a los que puede acceder. Por defecto, cada aplicación se ejecuta en su propio proceso Linux, independientemente del resto, teniendo cada proceso su propia maquina virtual. Cuando alguno de los componentes de una aplicación necesita ejecutarse Android inicia su proceso y no lo termina hasta que no lo necesita más o requiere la memoria para otras aplicaciones. Es posible permitir que las aplicaciones accedan a los ficheros de otras aplicaciones y a la información interna del dispositivo (contactos, mensajes, imágenes, cámara, etc.).

5.1 Componentes de las aplicaciones

Son los componentes básicos [1] a través de los que se construyen las aplicaciones Android. Cada componente es único y juega un papel específico en la definición del funcionamiento de la aplicación. Los componentes existen como entidades individuales, aunque puede depender de otros, teniendo su propia finalidad y ciclo de vida, que marca cuando son creados y destruidos. A continuación se muestran los cuatro tipos de componentes:

- **Actividades:** Representan una única pantalla de la interfaz de usuario. Por ejemplo, esta aplicación tiene una actividad para el menú principal con todas sus opciones, y para cada una de ellas, otra actividad. Aunque trabajen de forma conjunta, cada una de ellas es independiente del resto. Otra aplicación podría iniciar alguna de estas actividades siempre que tenga permisos. Es lo que ocurre por ejemplo cuando se accede a la cámara de fotos del dispositivo desde la pantalla de inventario. Todas las actividades están implementadas como una subclase de *Activity.java*.

- **Servicios:** Son componentes que se ejecutan en segundo plano para realizar tareas largas o con procesos remotos sin afectar a la interfaz de usuario. Por ejemplo, un servicio puede estar accediendo a las partidas abiertas que tiene un usuario sin impedir que este siga interactuando con la UI. Las actividades pueden lanzar estos servicios y dejarlos ejecutándose mientras sigue realizando sus tareas. Todos los servicios se implementan como subclases de *Service.java*.

- **Proveedor de contenidos:** Componente que se encarga de gestionar un conjunto compartido de datos de la aplicación. Puede almacenar los datos en el sistema, la base de datos, la web o en cualquier otro lugar donde se asegure el acceso a la aplicación. A través de un proveedor de contenidos otras aplicaciones pueden acceder o modificar la información (si este lo permite). No se usará para este proyecto.

- **Receptores de anuncios:** Componente encargado de responder a los anuncios transmitidos por todo el sistema. Muchas de las emisiones se originan en el propio sistema (pantalla apagada, batería baja, imagen captura,…) y pueden ser utilizadas y creadas desde la propia aplicación. Aunque no afectan directamente a la interfaz de usuario se pueden crear notificaciones para alertar al usuario de algunos eventos. Se implementan como subclases de *BroadcastReceiver.java* y cada emisión se entrega como un objeto Intención.

Una característica única de Android es que cualquier app puede iniciar componentes de otras aplicaciones. Por ejemplo, no se necesita añadir todo el código que activa la cámara para tomar una fotografía, simplemente se inicia la actividad encargada de ello y parece que la cámara forma parte de la aplicación.

Las actividades, los servicios y los receptores de difusión pueden activarse a partir de mensajes asíncronos denominados *intenciones*. Estos sirven para enlazar componentes independientes, en tiempo de ejecución, sin importar que sean de esta aplicación o de otra, como si se pidiera al otro componente que realizase una determinada acción. Para las actividades y los servicios determinan la acción a realizar y pueden incluir los datos a los que estas afectarán.

5.2 Fichero de manifiesto

Antes de que el sistema Android pueda iniciar ningún componente debe asegurar que esos componentes existen en la aplicación a partir del fichero *AndroidManifest.xml* en el que se deben declarar todos los componentes que forman la app. Además de la declaración de componentes, el manifiesto se encarga de dar permisos a la aplicación, declarar la versión mínima que debe tener el dispositivo, las características hardware y software que utilizará la aplicación, añadir las librerías que se quieren utilizar, entre otras. La estructura que mostrará el manifiesto del proyecto es la siguiente:

```
<manifest ... >
    ...
    <application android:icon="@drawable/ic_launcher" ...>
        <activity android:name=".Initial" ... >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        ...
    </application>
</manifest>
```

Ilustración 10. Estructura del manifiesto

Todos los componentes que se declaran en el manifiesto deben seguir la estructura <*activity*>, <*service*>, <*reciver*> ó <*provider*> a los que se les pueden incluir atributos como icono, nombre, etiqueta, etc. Se incluirán en las actividades <*intent-filter*> que especifican funcionalidades disponibles de la actividad para responder a intenciones de otras aplicaciones. En este caso, dado que no se espera recibir peticiones, se ha utilizado únicamente para especificar la actividad en la que se inicia la el juego.

Para el juego desarrollado se necesitarán permisos para añadir efectos de vibración y animación, para administrar las conexiones a internet y asegurar que los usuarios tienen todo lo necesario para poder disfrutar del juego. A continuación se muestra un ejemplo de cómo se especifican en el manifiesto estos permisos.

```
<uses-permission android:name="android.permission.VIBRATE" />
...
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

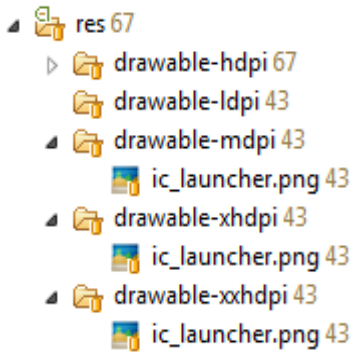
Ilustración 11. Permisos en el manifiesto

El manifiesto de este proyecto incluye la declaración de todas las pantallas/actividades que se muestran en la Ilustración 6 y las Ilustración 7, todos los servicios necesarios para la interacción con el servidor y todos los permisos necesarios para el funcionamiento del juego. No se utilizaran proveedores de contenido ni receptores de difusión.

5.3 Recursos en Android

Una aplicación Android está compuesta por muchas más cosas a parte del código. Requiere recursos separados del código fuente como son las imágenes, los sonidos y cualquier otra cosa relacionada con la presentación visual de la app. Utilizando cada tipo de recurso de forma independiente se consigue facilitar enormemente la actualización de todas las características de la aplicación ya que no se necesita modificar apenas código. Además permite optimizar la aplicación para la gran variedad de dispositivos de Android. Cada tipo de recurso se coloca en un subdirectorio específico dentro de la capeta res del proyecto.

Las imágenes se almacenan en las carpetas *drawable*. En el proyecto existen varias carpetas destinadas a contener las imágenes, todas ellas con el mismo nombre pero con una terminación diferente (hdpi, ldpi, mdpi, xhdpi y xxhdpi). Para cada imagen que se use dentro de la aplicación, existirá una copia de mayor o menor calidad, con exactamente el mismo nombre, dentro de cada una de esas carpeta. Según las características del dispositivo (tamaño de la pantalla, densidad y resolución), Android es capaz de determinar cuál es la imagen que mejor se adapta al dispositivo que utiliza la aplicación.



De la misma forma que Android escoge la imagen que más se adecue al dispositivo utilizado, también es capaz de seleccionar los ficheros de diseño que mejor se adaptarán a este. Para ello se utilizará un método de almacenamiento de los ficheros de diseño muy similar al de las imágenes. Este, además, será capaz de diferenciar los ficheros que están enfocados para una versión de los que lo están para otra. En la carpeta *values* se almacenarán ficheros XML con valores (texto, dimensiones, colores, arrays y estilos) que se utilizaran en la interfaz de usuario. Por ejemplo los ficheros contenidos en la carpeta *values-v11* serán utilizados por los dispositivos que tengan instalada una versión igual o inferior a esa.

En línea con lo anterior, cada carpeta values contiene los siguientes tipos de ficheros:

- **Strings.xml:** Se asigna a variables estáticas unas cadenas de caracteres que se quieren mostrar en el juego. Estas variables se referencian más adelante, desde el código, en el lugar donde se quiere mostrar texto. Mediante la modificación de este fichero se consigue cambiar los mensajes del juego sin la necesidad de buscar donde se incluyó el texto en el código. Además, esto permite crearse distintos ficheros *Strings.xml* con la terminación de un idioma (-en, -es, -fr), en los que los nombres de las variables se mantienen y simplemente se modifica el mensaje a mostrar. Dependiendo de la configuración de idioma del terminal que se use, Android cargará un fichero u otro, consiguiendo así que la aplicación se pueda traducir a muchos idiomas con un mínimo esfuerzo.

- **Colors.xml:** Igual que en el punto anterior se asignan valores hexadecimales que corresponden a colores a una serie de variables. Esta se referencian desde otros ficheros de diseño o desde el propio código y asignan ese color al objeto que se desea. Facilita enormemente la elección y modificación de los colores de la interfaz sin tener que modificar apenas código. Además se consigue que el código sea más comprensible al asignar la variable *"yellow"* a un botón en lugar de *"#FFFF00"*.

- **Dimens.xml:** Se utiliza de la misma forma que los anteriores pero permite almacenar valores de dimensiones que se utilizarán para tamaños, separaciones y otros aspectos de la interfaz. Es interesante observar, sin modificar el código y teniendo varios ficheros de dimensiones, como se adapta la aplicación a las dimensiones de los distintos dispositivos que utilizan Android.

- **Arrays.xml:** Permite almacenar arrays de valores que se pueden asignar a múltiples objetos que se utilizan en la interfaz. Un ejemplo de utilización de este fichero, en línea con el siguiente punto, son los desplegables que se muestran en el menú de preferencias de la aplicación para permitir al usuario elegir el estilo con el que quiere que se muestre la UI.

- **Styles.xml:** Son ficheros que especifican características de la UI durante el juego. Pueden almacenar características combinadas de los puntos anteriores y conseguir, por ejemplo, que todos los botones de la aplicación se muestran en el mismo color y forma.

Aunque no sean recursos como tales, la carpeta *assets* del proyecto contiene archivos que se utilizarán, pero que Android no es capaz de tratar como recursos. Esta carpeta contiene los scripts SQL de la base de datos interna del dispositivo y para utilizarlos ha sido necesario implementar ciertas funcionalidades que se explicaran más adelante.

Por último y probablemente lo más importante, están los ficheros de diseño de la aplicación. Incluidos en las carpetas de *layout*, se encuentran los ficheros que indican qué elementos contiene cada actividad y como están colocados dentro de ella. No tiene porque existir un único fichero de diseño XML para cada actividad. Por ejemplo en la actividad *Game_inventario.java* se dispone de un fichero para mostrar y colocar todos los elementos, entre los que se incluye un *ListView*, y otros para indicar como se debe mostrar cada elemento interno de la lista. Estos ficheros siguen todo el mismo patrón. Están formados por elementos de tipo *Layout*, que a su vez son contenedores de otros elementos que

pueden ser *Views*, como botones y desplegables, u otros *Layouts* [1]. A continuación se mostrará un ejemplo de una estructura que se consigue mediante estos ficheros y se detallarán los tipos algunos tipos de *layout* y *view* que han sido importantes en este proyecto.

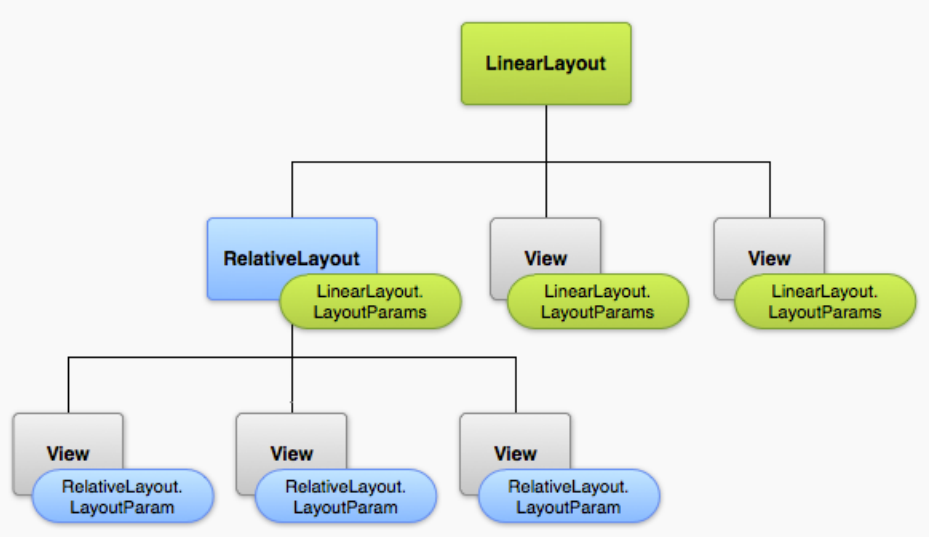


Ilustración 12. Estructura de ficheros de diseño

- **Linear Layout:** Permite organizar todos los elementos que contiene en una línea horizontal o vertical.

- **Relative Layout:** Permite organizar todos los elementos que contiene en función de las posiciones del resto de elementos que contiene y de sí mismo. Por ejemplo el objeto 2 puede estar a la izquierda del objeto 1 y justo encima del objeto 3. Cuando contiene muchos elementos resulta poco eficiente al almacenar mucha información.

- **List View:** Permite crear una lista con los elementos que contiene. Si se añaden más objetos de los que caben en su tamaño original añade un scrolling automático que permite que se sigan mostrando todos.

- **Grid View:** Su función es prácticamente la misma que la del List View pero los elementos que contiene se muestran en cuadricula. Dependiendo de la información que se desee mostrar será más conveniente uno u otro.

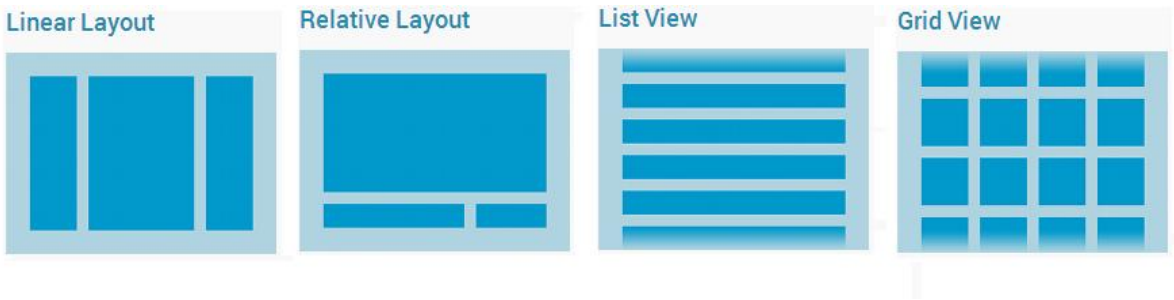


Ilustración 13. Tipos de Layout

Para conseguir un resultado óptimo y aparentemente bonito es muy recomendable utilizar una combinación de los layout antes citados. Existen muchos más tipos de layout como Grid Layout, Absolute Layout, Table Layout y Frame Layout pero estos han sido los que más se han utilizado en este proyecto.

Como ya se ha podido intuir los recursos son simplemente datos (cadenas, imágenes,…) que se almacenan fuera del código de la aplicación para mejorar su organización, garantizar su unicidad y permitir su adaptación a las características del dispositivo como, por ejemplo, el idioma, la orientación o el tamaño. Todos los recursos explicados, que se crean en la carpeta *res*, no pueden ser compartidos con otras aplicaciones pero si se puede acceder a ellos desde otros ficheros de recurso o desde el propio código java, como se muestra a continuación.

```
android:text="@[tipo_de_recurso]/[nombre_del_recurso]" />

setContentView(R.layout.activity_game_inventario);
```

Ilustración 14. Accesos a recursos

A continuación se explicaran los pasos más importantes que se han seguido para la implementación de la interfaz de usuario. Era necesario explicar cómo funciona y utiliza Android toda la información que se necesita para cualquier aplicación.

5.4 Interfaz de usuario

Cada una de las pantallas que forman una aplicación es básicamente una actividad. Cada actividad se implemente en una clase Java que hereda de la clase base *Activity* de Android. Las actividades interactúan con los usuarios a través de su interfaz de usuario. Esta se implementa a partir de objetos de las clases View y ViewGroup, que se pueden ver como vistas (botones, cuadros de texto, listas, …) y contenedores. Por ejemplo, la actividad *App_Menu.java* de este juego se compone de vistas de tipo Button y un contenedor de tipo LinearLayout que organiza su contenido en línea verticalmente. Aunque la implementación de la interfaz se ha realizado casi exclusivamente en fichero de diseño XML, también se puede realizar directamente desde el código java. Cada actividad tendrá sus propios ficheros de diseño independientes. Algunas actividades disponen de varios ficheros distintos que se utilizarán en función de las características de los dispositivos. En una pantalla más grande interesa poder mostrar más información.

Para definir el diseño que utilizará una actividad se usa *setContentView()*, que a partir del identificador del recurso es capaz de cargar las vistas de la pantalla. Aunque es posible crear el diseño directamente desde el código se ha preferido utilizar esta forma, ya que resulta más limpio. Para todas las actividades, la llamada a este método se ha realizado sobrescribiendo el método onCreate(), que se ejecuta cuando se crea la pantalla. En este nuevo método, que siempre debe llamar al método padre para no perder funcionalidad, se establecen los *Listeners* de las vistas y se preparan todas las variables que se usaran en el resto de métodos cuando es necesario. En algunos casos se ha sobrescrito también el método onResume() como se explicará más adelante. A continuación se muestra un claro ejemplo de cómo quedaría este método para la actividad *App_Menu*.

```
public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_app_menu);
    //Set listeners for app_menu buttons
    Button ButtonNewGame = (Button) findViewById(R.id.app_Menu_ButtonNewGame);
    Button ButtonJoinGame = (Button) findViewById(R.id.app_Menu_ButtonJoinGame);
    Button ButtonSavedGame = (Button) findViewById(R.id.app_Menu_ButtonSavedGames);
    Button ButtonExtras = (Button) findViewById(R.id.app_Menu_ButtonExtras);
    ButtonNewGame.setOnClickListener(this);
    ButtonJoinGame.setOnClickListener(this);
    ButtonSavedGame.setOnClickListener(this);
    ButtonExtras.setOnClickListener(this);
}
```

Ilustración 15. Ejemplo del método onCreate() sobrescrito.

Una vez preparados y asignados los diseños de cada actividad, es el momento de establecer las respuestas de la interfaz a las acciones que realizan los usuarios. Existen varias formas de llevar a cabo esta tarea:

- Asignando un método directamente a cada vista desde los Layouts XML.
- Haciendo que la actividad implemente la interfaz *onClickListener*.
- Implementando un escuchador de eventos y registrándolo en la vista.

Debido a los requerimientos de este proyecto, únicamente ha sido necesario utilizar las dos primeras opciones. En las actividades que no incluían demasiados elementos en su interfaz, no suponía ningún problema asignar en los ficheros de diseño un método para cada elemento. Sin embargo, para otras actividades con un mayor número de elementos o con listas dinámicas, ha resultado mucho más útil hacer que la propia actividad implemente un escuchador de eventos.

A partir de uno o varios métodos y de los identificadores de las vistas, la aplicación es capaz de saber qué es lo que tiene que ejecutar. A continuación se muestra un ejemplo del método *onClick()* que se sobrescribe de la interfaz *onClickListener*. Se puede ver que, dependiendo del identificador del elemento que pulse el usuario, la aplicación irá a una actividad o a otra.

```
@Override
public void onClick(View arg0) {

    if (arg0.getId() == R.id.app_Menu_ButtonNewGame)
        startActivity(new Intent(this, App_NewGame.class));
    else if (arg0.getId() == R.id.app_Menu_ButtonJoinGame)
        startActivity(new Intent(this, App_JoinGame.class));
    else if (arg0.getId() == R.id.app_Menu_ButtonSavedGames)
        startActivity(new Intent(this, App_SavedGames.class));
    else if (arg0.getId() == R.id.app_Menu_ButtonExtras)
        startActivity(new Intent(this, App_Extras.class));
}
```

Ilustración 16. Ejemplo implementación de onClickListener

Aunque el más utilizado sea el *onClick* existen otros métodos para gestionar la interacción del usuario con la interfaz, como por ejemplo:

- **onClick():** se ejecuta cuando el usuario o bien pulsa sobre el elemento con el control táctil, lo enfoca con la teclas de navegación y pulsa Enter.

- **onLongClick()**: se ejecuta cuando el usuario o bien pulsa y mantiene pulsado sobre el elemento con el control táctil, lo enfoca con la teclas de navegación y pulsa y mantiene el botón Enter.

- **onFocusChage()**: se ejecuta cuando el usuario navega hasta el elemento o sale de él utilizando las teclas de navegación.

- **onKey():** se ejecuta cuando el usuario enfoca el elemente y pulsa o suelta una tecla del dispositivo.

- **onTouch()**: se ejecuta cuando el usuario realiza cualquier acción sobre la pantalla del dispositivo, siempre que sea dentro de los límites del elemento.

Una vez que se muestran los elementos y estos responden a las acciones de los usuarios, se usaran *Intents* para realizar la navegación entre las distintas actividades. Como se puede observar en la Ilustración 16, utilizando el método *startActivity()*, que espera recibir un *Intent* que describa la actividad a la que se quiere ir, se puede cambiar de pantalla fácilmente. En este punto la aplicación ya muestra su esqueleto base y permite navegar entre las distintas pantallas que la forman.

Para seguir completando la interfaz se van a añadir los menús de opciones, diálogos y preferencias. Estos mostrarán gráficamente opciones que al ser seleccionadas, realizaran las tareas correspondientes.

5.4.1 Menú de opciones y diálogos

Se utilizará un único menú de opciones común para todas las actividades. Para ello, se ha creado una clase padre *Op_Menu.java*, que hereda de *Activity*, de la que heredan todas las demás clases. Los métodos necesarios para gestionar los menús se implementan en esta clase y se sigue manteniendo la herencia de *Activity* del resto de actividades. Con esto se evita duplicar código en cada actividad.

Para mostrar este menú de opciones en cada una de las actividades hay que implementar dos métodos en la actividad *Op_Menu*:

- **onCreateOptionMenu():** Este método se ejecuta cuando se pulsa el botón de menú del dispositivo. En él se debe inflar el fichero de diseño XML que contendrá las opciones de esta nueva interfaz de usuario.

- **onOptionsItemSelected():** Este método se ejecuta siempre que se selecciona uno de los elementos del menú de opciones creado anteriormente. El código interno del método debe identificar el elemento seleccionado y realizar la tarea correspondiente a dicho elemento.

El menú de opciones de esta aplicación contiene los métodos necesarios para acceder a las actividades de ajustes, perfil de usuario, acerca de y ayuda, además de una opción que permite a los usuarios abandonar su sesión. Las pantallas de perfil de usuario y de ayuda son actividades normales que se gestionan como ya se ha explicado, mientras que la opción *Acerca de*, por el contrario, se ha decidido mostrar como un dialogo, que no tapa completamente la actividad en la que se encontraba el usuario. Para ello, en su declaración en el manifiesto, se le ha añadido un tema que permite mostrarla de esta forma. Los ajustes se explicarán más adelante en el apartado de Preferencias.

```
<activity
    android:name=".Op_About"
    android:label="@string/op_about_activity"
    android:screenOrientation="landscape"
    android:theme="@android:style/Theme.Holo.Dialog" >
</activity>
```

Ilustración 17. Ejemplo tema para diálogo en el manifiesto

Para la parte de diálogos, o mensajes que se muestran al usuario, se han utilizado dos métodos diferentes para mostrar la información a los usuarios:

- **Toast:** Simplemente muestran un pequeño mensaje, que se desvanece automáticamente al tiempo (dependiendo de los parámetros que reciba). Estos se han utilizado para informar al usuario sobre las acciones que realiza en el juego, como por ejemplo, objeto añadido al inventario, registro realizado con éxito, partida creada, campo contraseña debe completarse, etc. A continuación se muestra un ejemplo de cómo lanzar uno de estos mensajes. La función *makeTest()* recibe como parámetros el contexto de la actividad, el mensaje y el tiempo que permanecerá en la pantalla. Este tipo de mensajes permiten que el usuario siga teniendo el foco en la actividad y pueda seguir interactuando con ella, aunque se esté mostrando la información.

```
Toast.makeText(this, R.string.app_login_usernamevacio, Toast.LENGTH_LONG).show();
```

- **AlertDialog:** Este tipo de mensajes implican una acción de respuesta del usuario. Aunque se han utilizado para mensajes de confirmación, que muestran el texto y el usuario debe confirmar o cancelar la acción, también existe la posibilidad de añadirles opciones, listas desplegables y campos de texto. A continuación se muestra un ejemplo del mensaje de fin de partida.

```
public void juegoTerminado() {

    new AlertDialog.Builder(this)
            .setTitle(R.string.gameOverTitle)
            .setMessage(R.string.gameOverMessage)
            .setPositiveButton(R.string.gameOverPositiva, new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    volverAJugar();
                }
            })
            .setNegativeButton(R.string.gameOverNegativa, new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    alMenuPrincipal();
                }
            }).show();
}
```

Ilustración 18. Ejemplo mensaje AlertDialog()

5.4.2 Preferencias

Android dispone de un mecanismo para almacenar pequeñas cantidades de datos en forma de pares clave/valor en ficheros XML. La información que se guarda utilizando este formato no se pierde aunque se detenga la aplicación o se apague el dispositivo. Esto se conoce como almacenamiento persistente. Los datos almacenados pueden ser privados de la actividad o pueden compartirse en toda la aplicación. Se utilizará este tipo de almacenamiento para guardar las preferencias de los jugadores, o ajustes. Solo se pueden almacenar datos que se puedan identificar como un *String* y tengan un valor de los siguientes tipos: *Boolean*, *Integer*, *Long*, *Float* o String.

Para comenzar, hay que crearse una clase, que llamaremos *All_PreferenceFragment*, que herede de *PreferenceFragment*, que ya contiene todos los métodos que utiliza Android para gestionar este tipo de información. Como ya se hiciera anteriormente, hay que sobrescribir *onCreate()* para añadir la llamada al método *addPreferencesFromResource()*, en el que se indicará el fichero XML en el que se ha definido como va a ser la pantalla de preferencias. Este fichero no es exactamente igual que los utilizados para los layout de las actividades normales, la estructura es muy similar pero se utilizan elementos propios de las preferencias (*CheckBoxPreference*, *ListPreference*,...). Esta clase se encargará de mostrar, modificar y guardar las opciones que elija el usuario. A continuación se crea una actividad normal, que llamaremos *All_Preference*, para la que también se sobrescribe el evento *onCreate()*, pero que no carga ningún fichero de diseño (en realidad carga uno vacio) y en su lugar, utilizará el fragmento creado anteriormente.

Mediante la actividad de preferencias y el fichero de diseño del fragmento de preferencias se especificarán la relación entre los elementos de la interfaz y los pares de clave/valor que almacenarán la información. En la actividad se declaran los nombres de las claves y sus valores por defecto y luego se utilizan en fichero XML.

Para acceder a las preferencias compartidas por todas las actividades, hay que utilizar el método *getSharedPreferences()* y especificar el fichero XML de preferencias y el nombre de la preferencia. Se utilizará un único fichero de preferencias (el que viene por defecto) y se declararán los métodos de acceso y modificación en la misma actividad de preferencias. Para acceder a los valores guardados en las preferencias se utilizan los métodos de *SharedPrefences getBoolean()*, *getString()*, … dependiendo del tipo de dato que se desee, y para modificarlos, los métodos *setBoolean()*, *setString()*, … de la misma forma.

Una vez que la aplicación es capaz de almacenar todas las preferencias definidas, estas podrán utilizarse en el resto de la aplicación. Se sobrescribirá el evento *onResume()* de todas las actividades que tengan funcionalidad que dependa de estos valores. Por ejemplo, la opción de los efectos de sonido viene activada por defecto, pero el usuario desea deshabilitarla y accede a los ajustes desde la actividad del menú principal. Esta actividad quedara en segundo plano mientras el usuario permanezca en los ajustes y cuando regrese ejecutará el método *onResume()*, para pasar del estado de pausa al de activa. En esa llamada se comprobará el valor de la preferencia con clave *key_sonido*, y al encontrarse desactivada, la aplicación deberá parar la música. Lo mismo ocurrirá para el resto de opciones del menú de preferencias.

Esta forma de almacenar la información resulta muy útil para gestionar las sesiones de los usuarios. Una vez que un usuario inicie sesión, esta permanecerá activa, aunque se cierre la aplicación o se apague el dispositivo. De esta forma se mantendrá toda su información disponible y no tendrá que andar iniciando sesión cada vez que inicie la aplicación.

5.4.3 Puntos interesantes

En este apartado se comentarán algunos aspectos de la interfaz de usuario del proyecto que han resultado interesantes. Debido a los nuevos requerimientos del cliente, se han realizado modificaciones en algunos de los elementos de la interfaz. A continuación se explicaran brevemente en que han consistido estas modificaciones.

Listas de elementos

Para algunas de las listas de elementos que se incluyen en la parte del juego se ha tenido que modificar la forma en que se muestran los elementos. El cliente insistió en que, tanto los objetos que se añaden al inventario, como los disquete que se añaden a la lista de programas, debían ser considerados de distinta forma dependiendo del su tipo. Los disquetes podían ser de tipo ofensivo o defensivo, y los objetos de tipo básico o complejo. En ambos casos, dependiendo del tipo de objeto, los elementos debían mostrar una información u otra en la lista. Para explicar las modificaciones que esto supuso, se utilizará de ejemplo el inventario de objetos, pero el proceso es el mismo para la lista de programas.

El layout de la actividad *Game_Inventario* se mantiene exactamente igual, no se tuvieron que realizar modificaciones en este aspecto. Sin embargo, antes los elemente que se mostraba en la lista disponían de un único layout, que era el que se cargaba sin importar el tipo de objeto, pero ahora, como hay que diferenciar entre objetos básicos y compuestos, hay que disponer de dos layouts diferentes. El layout de los objetos básicos es acumulable, es decir, si el jugador tiene 3 trozos de metal en el inventario, solo se debe mostrar un elemento en el que se indique la cantidad. Por el contrario, los objetos compuestos no deben acumularse. Si el jugador tiene 2 pistolas en su inventario, se deben mostrar ambas independientemente, ya que, aunque se trate del mismo objeto, las características de ambos pueden ser diferentes durante la partida (los objetos pierden durabilidad y su creación depende de varios factores).

El inventario de cada jugador es un *HashMap<String, Arraylist<Item>>*, en el que existirá una clave para cada objeto y un *ArrayList* que almacenará todos los objetos con es clave que se tienen. Para los objetos simples, como las hiervas, solo se almacena un elemento en el array, y se le va modificando su atributo cantidad según el jugador añada más o use las que tiene. Para los objetos combinados, se almacenan tantas instancias del objeto como se tengan (Una clave para la pistola, tantas pistolas como se tengan en el array). Debido a esta forma de almacenamiento, ha sido necesaria la implementación de dos clases nuevas.

- **SmartAdapter:** Que se encargará de identificar el tipo del objeto y asegurar que se le asigna correctamente su layout.

- **Rango:** Una clase intermedia necesaria para el SmartAdapter.

La clase *SmartAdapter* hereda de la clase base *BaseAdapter*, que es la que utiliza Android para gestionar los elementos de las listas. Esta clase cuenta con una serie de métodos por defecto, que utiliza para mostrar los elementos correctamente. Debido a las modificaciones realizadas, estos métodos ya no son válidos y es necesario modificarlos en nuestra clase. La función *getCount()* del *BaseAdapter* se encarga de notificar a la lista cuantos elementos tiene, pero como se ha añadido un *ArrayList* para cada clave del *HashMap* y los objetos se añaden de diferente forma dependiendo de su tipo, no es capaz de realizar su tarea. Para solucionar esto se ha implementado un sistema de rangos. Este sistema permite traducir nuestra implementación a algo que el BaseAdapter pueda utilizar. Para cada clave del HashMap se crea un Rango acumulativo, en una lista de rangos, con el número de elementos que el array contiene (siendo el rango inferior, el superior del anterior y el superior, el inferior mas el número de elementos del array). Tras analizar todo el HashMap se modifica el getCount() para que devuelva el rango superior del último rango de la lista, es decir el número de elementos que se mostraran en el inventario.

La función *getView()* recibe tantas llamadas, con la posición de cada elemento, como devuelve el getCount() y se encarga de prepara la vista de cada uno de ellos. En esta función hay que traducir la posición que recibe al elemento del HashMap que realmente es. Para ello se utilizan de nuevo los rangos. A continuación se muestra una imagen de ejemplo de este proceso.

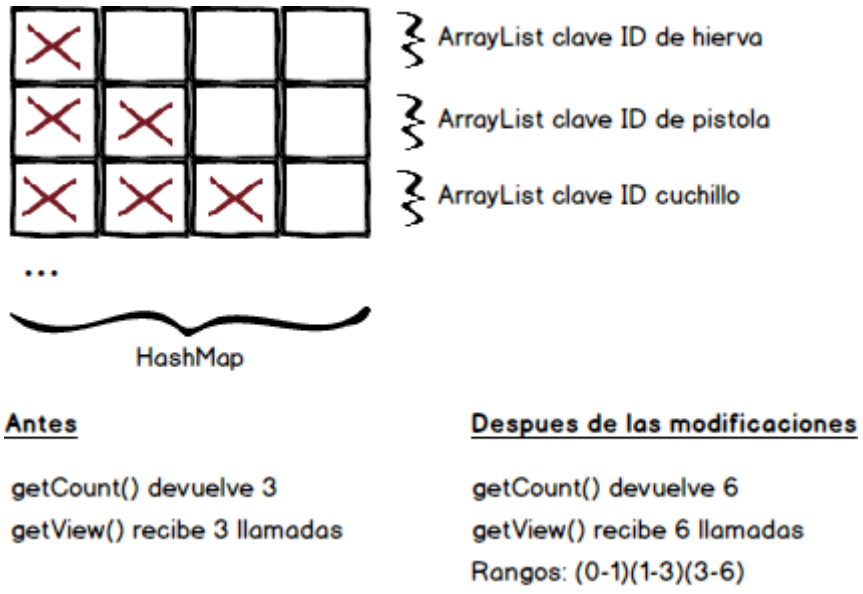


Ilustración 19. Modificaciones SmartAdapter

Si a *getView()* le llega la posición 6 en su última llamada, la función deberá traducirla para que sea capaz de acceder al elemento dentro del *ArrayList* que se encuentra en el *HashMap*. Buscando el rango al que pertenece la posición se saca el ArrayList en el que se encuentra el objeto, y a partir del valor de la posición dentro del rango, la función es capaz de identificar el objeto exacto que debe tratar. Ya solo le faltará identificar el tipo de objeto y asignarle a la vista que le corresponda.

El SmartAdapter no recibe el HashMap del inventario como tal, sino que recibe una colección con los valores que contiene. Esto permite a todas sus funciones trabajar fácilmente con índices.

Diseño fijo para el juego

Como ya se indicó en la Introducción a la aplicación, el dispositivo móvil que usa la aplicación simboliza un elemento real dentro del juego. El cliente desea que el diseño de la aplicación mantenga esta misma idea. El dispositivo que los personajes del juego llevan implantado tiene cinco botones de selección y dos de direccionamiento. Por esto se ha decidido que la interfaz de las pantallas de la aplicación, cuando se está en partida, deben mantener la misma estructura. A continuación se muestra una ilustración de ejemplo.

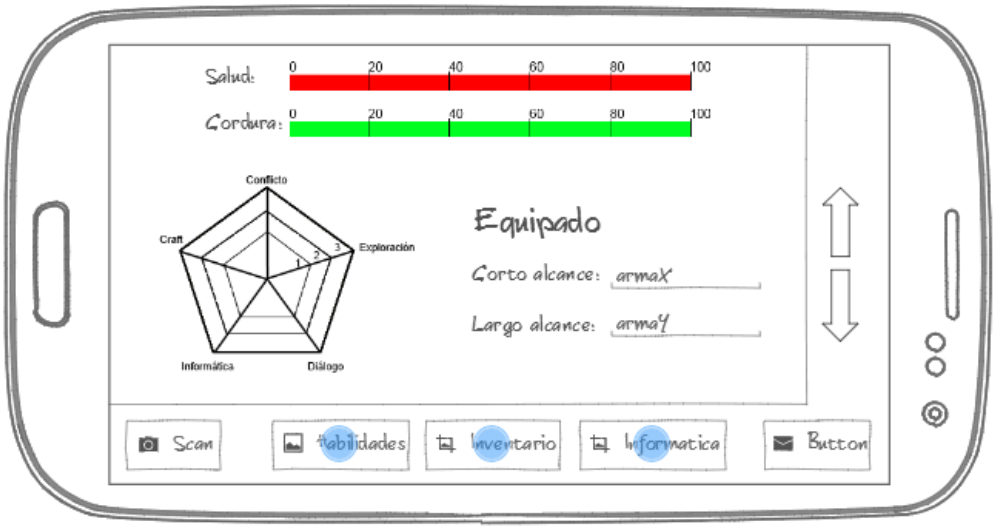


Ilustración 20. Ejemplo diseño de interfaz fijo

Los botones inferiores se utilizarán para poder navegar entre todas las pantallas de la interfaz del jugador, mientras que los botones laterales, las flechas que se pueden ver en el diseño anterior, se utilizarán para que el jugador pueda deslizar las listas disponibles en algunas de las interfaces.

5.5 Base de datos interna

Android utiliza el motor de base de datos SQLite que no precisa de un servidor. La base de datos es un simple fichero almacenado en el sistema de archivos. Para mejorar el código de la aplicación se ha decidido encapsular toda la funcionalidad de la base de datos en una clase Java intermedia denominada *DatabaseAdapter*. En su interior se define otra clase privada *DatabaseHelper* que hereda de *SQLiteOpenHelper*. Esta clase utiliza los métodos *onCreate()* y *onUpgrade()* para crear y actualizar la base de datos cuando sea necesario.

La clase DatabaseHelper llama al método *onCreate()* solo la primera vez que se accede a la base de datos. Este método se encarga de cargar, leer, interpretar y ejecutar los scripts SQL que crean la base de datos y cargan la información en ella. Por el contrario el método *onUpgrade()* se ejecuta automáticamente cada vez que la versión con la que se guardo la base de datos no coincide con la actual (variable global *DATABASE_VERSION*). Existen diferentes formas de implementar la actualización de la bbdd pero para este proyecto, que no cuenta con demasiadas tablas ni datos, se ha decidido que cada vez que se actualice la bbdd, esta será borrada completamente y creada de nuevo, al igual que se hacía en su creación normal.

Una vez que ya se tiene la bbdd creada y que se puede actualizar cuando se desee, se puede proceder con la implementación de los métodos de acceso. Dentro de la clase *DatabaseAdapter* se incluyen dos métodos más open() y close(), que permiten acceder y modificar los datos de la bbdd y que deberán ser llamado antes y después de realizar cualquier consulta. Dentro de la misma se incluyen todos los métodos necesarios para el desarrollo de juego y que utilizan la base de datos de alguna forma. Algunos ejemplos de estos métodos son: *findEventByID()*, *findIngradientsByID()* o *findCraftableItems()*.

Para la lectura de los scripts SQL se ha implementado en el proyecto la clase *SQLparser*, para que analice gramaticalmente el contenido de estos antes de poder ejecutarlos. Android cuenta con una función *execSQL()* que permite ejecutar instrucciones de una en una. Para poder ejecutar todo el script, se ha implementado un método *execSQLFile()* en el parser, que divide todos los scripts en instrucciones independientes que se puedan ejecutar desde la función anterior. Para ello, identifica los finales instrucción y se ignoran todos los comentarios y separadores que puedan perjudicar a este proceso.

5.6 Servidor

Se ha utilizado un servidor LAMP para el desarrollo del servidor. Este tipo de servidores utilizan una combinación de Linux, Apache, MySQL y PHP. Por el momento, el servidor funcionará únicamente en local, y aunque, sus funcionalidades no serán integradas en la aplicación, se explicará el proceso a seguir para utilizar los scripts PHP desde el juego y estos serán probados desde la web.

Ilustración 21. Ejemplo de prueba de script

Para facilitar la gestión de la base de datos se utilizará PhpMyAdmin, una herramienta que permite realizar múltiples operaciones MySQL a través de una interfaz de usuario, como se muestra en la siguiente imagen.

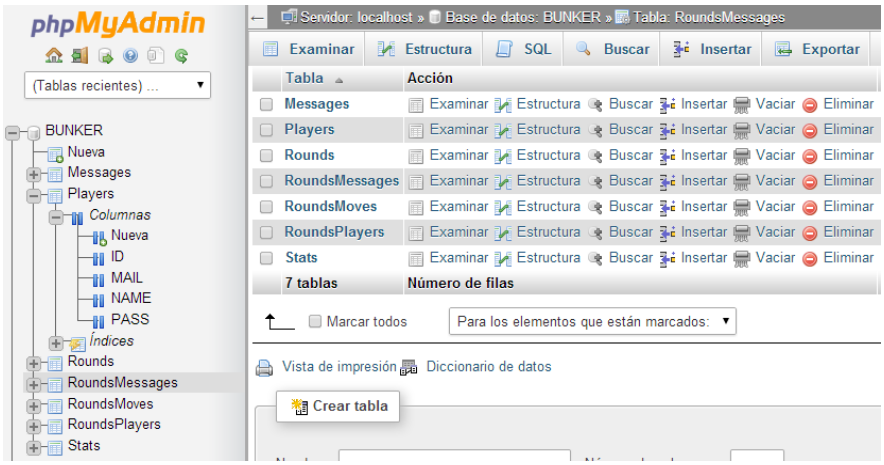


Ilustración 22. Base de datos desde phpMyAdmin

La estructura de la base de datos del servidor ya se explicó en el apartado 4.2.2 Diseño del servidor, pero a continuación se entra un poco más en detalle en las características de cada tabla y como las utilizará la aplicación.

Como tabla base del servidor se encuentra la tabla *Players*. Su clave primaria *ID*, única para cada usuario, identifica inequívocamente a cada jugador, y se utilizará durante el resto del juego para guardar la sesión de usuarios y gestionar la información que les afecta. Los atributos *name* y *pass*, que también son capaces de identificar a los usuarios, se utilizarán para realizar el inicio de sesión y el registro. Estas funciones devuelven siempre el identificador del jugador, que será el que se siga utilizando en la aplicación, o un valor erróneo que será tratado. Además, cuenta con información opcional que el usuario puede incluir, o no, en su registro.

Cada usuario registrado cuenta con unas estadísticas personales que se irán actualizando durante el juego. Estarán almacenadas en la tabla *Stats* y se identificarán a partir del *ID* del jugador. Las estadísticas incluyen información de las partidas como: partidas jugadas, partidas ganadas, combates realizados, etc.

En la tabla *Rounds* se almacenarán todas las partidas del juego. Cada partida incluye su propio identificador, que se usará para relacionarla con los jugadores que participan en ella y los movimientos y mensajes que realizan y envían. Para garantizar que la aplicación las gestiona debidamente, cuenta con registros de tiempo que indican en qué momento fueron creadas, cuando se realizó el último cambio y cuando se realizó el último movimiento. Además, para asegurar que se pueden guardar y retomar las partidas, la tabla almacena el turno actual, el número de jugadores y el estado. El estado cuenta con información relevante para permitir a los jugadores crear, unirse y abandonar las partidas. Una partida puede encontrarse *Abierta*, y permitir que los usuarios se unan, *Activa*, para indicar que se está jugando, *Guardada*, para indicar que se encuentra pausada y lista para retomar, o *Terminada*, momento en el que los jugadores no podrán interactuar mas con ella. Las partidas abiertas son las que se mostrarán a los jugadores que entren en la pantalla *JoinGame* y las guardadas a los que se encuentren en la pantalla *SavedGames*.

Como ya se ha comentado, a partir del identificador de la partida y el de los jugadores, se ha creado una tabla intermedia *RoundsPlayers* que contendrá los jugadores que participan en cada partida, junto con su estado dentro de la misma y su turno. Dentro del estado se almacenará toda su información de la partida, inventarios, salud, posición, etc.

Para la transmisión de mensajes dentro del juego también se utiliza la base de datos. Estos mensajes se guardarán en la tabla *Messages* y contendrán, aparte del texto a transmitir, los identificadores de los jugadores que envían y reciben el mensaje, la fecha de envío y un indicador para comprobar si el mensaje fue recibido o no. Para relacionar los mensajes con las partidas, se ha creado otra tabla intermedia *RoundsMessages*, que incluye las claves primarias de ambas tablas.

Por último, y para asegurar que la gestión de turnos dentro de cada partida es correcta, se almacenan todos los movimientos que realizan los jugadores en la tabla *RoundsMoves*. Esta tabla relaciona los jugadores con los movimientos que realizan en las partidas a partir de sus claves primarias, y almacena información sobre cada movimiento en cuestión.

Para la utilización de los scripts PHP del servidor en la aplicación, Android proporciona la clase base *AsyncTask* [18], que permite gestionar la ejecución de tareas en segundo plano mucho más fácilmente que creando hilos a mano. La primera tarea a realizar es crear una clase que herede de AsyncTask, que se utilizará para especificar cómo se realizará el acceso al servidor, la información que recibirá el usuario durante este proceso y cómo se devolverán los resultados a la aplicación. Para ello, se deberán sobrescribir en esta nueva clase los siguientes métodos:

- **onPreExecute():** Siempre se ejecuta justo antes que el código principal de la tarea asíncrona y se suele utilizar para preparar su ejecución e inicializar la interfaz.

- **doInBackground():** Ejecutará el código principal de la tarea. Por ejemplo el inicio de sesión de un usuario.

- **onProgressUpdate():** Ejecutará cada vez que se llame al método *publishProgress()* desde el método *doInBackground()*. Permite ir actualizando la interfaz de usuario antes que de que se reciba toda la información.

- **onPostExecute():** Se ejecuta cuando se termina de ejecutar la tarea asíncrona del método *doInBackground()*.

- **onCancelled():** Se ejecuta cuando se cancela la ejecución de la tarea asíncrona antes de su finalización.

El método doInBackground() se ejecuta siempre en un hilo secundario, y por tanto, no se puede interactuar con la interfaz desde el. Sin embargo, como todos los demás métodos se ejecutan en el hilo principal, si permiten realizar acciones sobre la interfaz. Como entre ellos se encuentra *onProgressUpdate*, y este puede ser invocado desde *doInBackground*, todos los métodos ofrecen la posibilidad de alterar la interfaz de una forma u otra.

Al crear una nueva clase que extiende de *AsyncTask* se deben indicar tres parámetros:

- **Parámetros:** Debe incluir todos los datos que la tarea necesita recibir. Para el ejemplo del inicio de sesión recibirá dos objetos de tipo *String*.

- **Progreso:** Debe incluir el tipo de dato que permitirá actualizar la interfaz mediante los métodos *onProgressUpdate* y *publishProgress*.

- **Resultado:** debe especificar el tipo de dato que devolverá al final de la tarea el método *doInBackground* y que recibirá el *onPostExecute*.

Para el ejemplo de la ejecución del script *Login.php* se espera que la clase AsyncTask tenga parámetros de tipo String (el nombre y la contraseña del usuario), no necesitará ningún parámetro de progreso (ya que no va a mostrar información intermedia) y que devuelva un *Boolean*.

```
public class TaskAccount extends AsyncTask<String, String, Boolean> {
```

Ilustración 23. Clase que extiende de AsyncTask

Cuando se ejecute el método *onPreExecute()*, se lanzará un diálogo de progreso que indicará al usuario que se está realizando la operación y le permitirá cancelarla si desea.

```
protected void onPreExecute() {

    waitDialog = ProgressDialog.show(context, "Please, wait", "Checking if username is available", true, true);
    waitDialog.setOnCancelListener(new OnCancelListener() {
        @Override
        public void onCancel(DialogInterface arg0) {
            Log.d(DEBUG_TAG, "onCancel inside onPreExecute()");
            TaskAccount.this.cancel(true);
        }
    });
}
```

Ilustración 24. Ejemplo del método onPreExecute

Seguidamente se ejecutará el método *doInBackground()*, que construirá la cadena de acceso al fichero PHP, dentro de la ruta del servidor, a partir de los parámetros *String* recibidos. Una vez se reciba una respuesta del servidor, este método se encargará de almacenar la información que sea necesaria y devolverá un dato de tipo Boolean.

```
@Override
protected Boolean doInBackground(String... params) {

    boolean result = false;
    if (MyPreference.getOnline(context))
        result = postSettingsToServer(params[0], params[1]);
    return result;
}
```

Ilustración 25. Ejemplo del método doInBackground

Este, será recibido por el método *onPostExecute()*, que cerrará el dialogo que se había creado en *onPreExecute()* y mostrará un mensaje informativo al usuario. También lo recibirá la actividad desde la que se ejecuto la tarea.

```
protected void onPostExecute(Boolean result) {
    if (!isCancelled())
        Log.d("DEBUG_TAG", "NewAccount task complete.");
    else
        Log.d(DEBUG_TAG, "Inside onPostExecute(), but cancelled.");
    waitDialog.dismiss();
}
```

Ilustración 26. Ejemplo del método onPostExecute

A continuación se muestra un ejemplo de cómo ejecutar una tarea en segundo plano desde una actividad.

```
String username = usernameEditText.getText().toString();
String password = passwordEditText.getText().toString();
boolean estaRegistrado = false;
TaskAccount account = new TaskAccount(this);
estaRegistrado = account.execute(username, password).get();
```

Ilustración 27. Ejemplo ejecución de una tarea asyncrona

6 Pruebas y resultados

En este apartado del proyecto se documentará el plan de pruebas realizado durante y tras el desarrollo de la aplicación, así como los resultados que se fueron recogiendo y sus consecuencias. El objetivo principal de esta fase es el de crear un subconjunto de todos los casos de prueba que tienen mayor probabilidad de detectar el mayor número posible de errores. Se persigue asegurar el correcto funcionamiento de la aplicación y corroborar que concuerda con todo lo especificado en el capítulo *3. Análisis de requisitos.*

Se realizarán las pruebas desde dentro hacia fuera, comenzando por los módulos unitarios para concluir con el sistema completo. Los tipos de pruebas que se han realizado para este proyecto son los siguientes:

1. **Pruebas unitarias:** Comprueba la lógica, funcionalidad y si es correcta la especificación de cada módulo. Se han llevado a cabo sobre todo durante la fase de implementación.

2. **Pruebas de integración:** Tienen en cuenta la agrupación de los módulos y el flujo de la información entre las interfaces. También realizadas casi exclusivamente en la fase de implementación.

3. **Pruebas de sistema:** Comprobar la integración con su entorno hardware y software. Realizadas constantemente ya que el código se ejecutaba, debugeaba y probaba desde dispositivos reales.

4. **Pruebas de validación:** Consisten en comprobar la concordancia con respecto a la especificación de los requisitos de la aplicación. Normalmente llevadas a cabo al terminar cada iteración de la fase de implementación.

5. **Pruebas de aceptación:** Asegurar que el producto se ajusta a los que el usuario quiere. Llevadas a cabo con los productos parciales resultantes de cada iteración de la fase de implementación.

Para una mejor comprensión de las pruebas que se mostrarán a continuación se van a describir algunos conceptos importantes sobre la estrategia de pruebas que se ha seguido en este proyecto:

- **Pruebas de caja blanca:** Se conoce el código fuente y se diseñan las pruebas específicamente para ese código, si este sufriera alguna modificación, su plan de pruebas tendría que ser rediseñado. Requieren mayores conocimientos técnicos y tiempo.

- **Pruebas de caja negra:** Se conoce la funcionalidad esperada del método pero no el código. Solo se proporcionan las entradas y salidas para verificar si son correctas, sin preocuparse de su funcionalidad interna.

A continuación se describirán cada uno de los tipos de pruebas de forma independiente y se mostrarán algunos ejemplos de cada uno de ellos.

6.1 Pruebas unitarias

Se trata de las primeras pruebas que se han realizado sobre el proyecto. Se realizan en la fase de implementación debido a que los costes para encontrar y subsanar errores en esta fase son bastante menores que en las siguientes. Las pruebas unitarias consisten en ir probando pequeños fragmentos de código denominados módulos, y a medida que se vaya avanzando ir creando nuevos módulo que contengan a los anteriores. Estas pruebas facilitan enormemente la localización de los errores en los nuevos módulos ya que los que lo forman ya han sido probados. Algunos ejemplos de pruebas unitarias realizadas para este proyecto son:

- **Prueba01:** Creación y actualización de la base de datos interna del dispositivo.

  En la clase *sqlParser.java* se incluyen los métodos que se utilizan para la creación y actualización de la base de datos a partir de dos scripts SQL que deben ser leído e interpretados antes de su ejecución. Para verificar que el parser funciona correctamente se realizaron las siguientes comprobaciones:

  - Se leen correctamente los scripts SQL.
  - Se ignoran los comentarios del código incluidos en los scripts.
  - Se dividen los scripts en las instrucciones independientes que estén en el formato adecuado para ejecutarse.
  - Se ejecutan las instrucciones y los resultados son correctos.
  - La ejecución completa de los scripts termina satisfactoriamente.

- **Prueba02:** Los botones de cada actividad funcionan perfectamente.

  Para todas las actividades que contienen botones existe un método que controla cuando se pulsa alguno de ellos. Para verificar que el *listener* funciona como se desea se realizaron las siguientes comprobaciones:

  - Se confirma que los botones responden cuando son pulsados.
  - Se comprueba que las acciones que realiza cada botón son las adecuadas.
  - Se prueba que los botones estén bloqueados o no se muestren cuando las condiciones así lo requieran.

- **Prueba03:** Mostrar correctamente el inventario.

  Cuando se añade un nuevo objeto al inventario la aplicación debe reconocer el tipo de objeto y guardarlo en el inventario como es debido. Para asegurar que la lista que simboliza el inventario se muestra y actualiza correctamente se han realizado las siguientes acciones:

  - Los objetos se añaden correctamente al inventario.
  - Los objetos de tipo básico se van acumulando en el inventario mientras que el resto aparece tantas veces como objetos se tengan.
  - Los objetos se eliminan del inventario cuando se utilizan o combinan sin volver a cargar la pantalla.

6.2 Pruebas de integración

Como ya se indicó anteriormente, consiste en combinar todos los módulos que ya fueron analizados en las pruebas unitarias, centrándose en probar únicamente sus interfaces (con un enfoque de caja negra). El primer paso es seguir en este tipo de pruebas es determinar cómo se combinan los distintos módulos. Para este proyecto se utilizarán pruebas de integración ascendentes, empezando por los módulos terminales de la aplicación y escalando hasta los superiores en la jerarquía de control. Se utilizarán módulos conductores para coordinar las entradas y salidas de cada caso de prueba. Una vez terminadas, serán eliminados para seguir combinando con los módulos superiores. Algunas pruebas relevantes para este proyecto han sido las siguientes:

- **Prueba01:** Accesos al menú de opciones.

  Los métodos para poder acceder al menú de opciones de la aplicación y todas las acciones a realizar desde este se han implementado independientemente en la clase *op_menu.java* y es necesario probar estos métodos desde las actividades en la que se permiten realizar estas operaciones:

  - Acceso a las preferencias y al perfil de usuario desde todas las actividades.
  - Comprobar que no se pierde información al acceder a las pantallas del menú de opciones *op_about.java* y *op_help.java,* que dejan la actividad principal en segundo plano.
  - Modificaciones en las preferencias que persistan en la aplicación.
  - Abandonar la sesión desde cualquier actividad.

- **Prueba02:** Gestión del inventario.

  Existen varios módulos que influyen directamente en cómo se muestran los objetos del inventario. Para asegurar el correcto funcionamiento de todas estas funciones conjuntamente se pueden destacar las siguientes comprobaciones:

  - Cuando se añade un objeto mediante su código QR, una combinación o un identificador, este se muestra en el inventario inmediatamente.
  - Cuando se utiliza o destruye un objeto, este desaparece del inventario o se reduce la cantidad, dependiendo de su tipo.
  - Cuando se equipa un objeto debe indicarse en el propio objeto y desaparecer del objeto equipado anteriormente.

- **Prueba03:** Listado de partidas.

  Todos los métodos de acceso al servidor relacionados con las partidas se utilizarán conjuntamente para garantizar que los usuarios puedan administrar sus partidas correctamente. Algunas de las comprobaciones realizadas han sido:

  - Cuando se crea una partida esta aparecen en la lista de *app_joingame.java*.
  - Cuando se borra una partida esta desaparece de la lista.
  - No aparecen partidas completas, empezadas, ni a las que ya te has unido.

6.3 Pruebas de sistema

Como ya se indicó anteriormente, este tipo de pruebas han sido constantes durante toda la implementación de la aplicación. Todo el código que se ha producido se ha ido ejecutando y debugueando desde una máquina virtual de Android, facilitada por el kit de desarrollo de Eclipse, o en dispositivos Android reales:

- **Prueba01:** *Sony Xperia U* con la versión 4.0.
    - Se ha probado la aplicación con este terminal de 3'5 pulgadas y con una resolución de 480x854 pixeles.
- **Prueba02:** *Samsung Galaxy Ace* 2 con la versión 4.1.
    - Se ha probado la aplicación con este terminal de 3'8 pulgadas y con una resolución de 480x800 pixeles.
- **Prueba02:** *Samsung Galaxy Note 10.1* con la versión 4.4.
    - Se ha probado la aplicación con este terminal de 10'1 pulgadas y con una resolución de 800x1280 pixeles.

Las pruebas de cliente se han realizado sobre estos mismos dispositivos. Dado que con la maquina virtual se puede simular casi cualquier dispositivo (con las características [10] que sean) y que con los terminales utilizados se abarcan casi todas las versiones para las que se asegura el funcionamiento de la aplicación, podemos confirmar el correcto funcionamiento del juego en cualquier entorno hardware y software.

6.4 Pruebas de validación

Consisten en comprobar que el producto que se muestra al cliente cumple las necesidades que se fijaron en el análisis de requisitos. Dado el tiempo de realización de este proyecto, no va a existir un producto final del juego como tal. El objetivo es poder desarrollar todas las funcionalidades principales para que el cliente pueda intuir cómo sería el resultado final. Debido a esto, se realizarán numerosas entregas intermedias, para asegurar que se van cumpliendo sus requisitos y para que pueda probar y opinar sobre los resultados. La técnica a utilizar es caja negra. Este proceso de pruebas constará de dos partes para cada entregable:

- Pruebas llevadas a cabo por el cliente en el entorno controlado de desarrollo.
- Pruebas realizadas por el cliente en su entorno de trabajo y sin observadores.

Al realizarse estas pruebas y las anteriores para cada entregable, se asegura que las partes de la aplicación que se van desarrollando son correctas y cumplen con los requerimientos del cliente.

6.5 Pruebas de aceptación

Es el último paso de la fase de pruebas antes de la entrega formal del software al cliente y se realiza, normalmente, en el entorno del usuario. Consiste en la aceptación por parte del cliente del software desarrollado, comprobando que el sistema está listo para su uso. Debido al tiempo de desarrollo del proyecto no va a existir un producto final y por tanto estas pruebas se dejarán como trabajo a realizar en el futuro.

6.6 Resultados

Los resultados obtenidos durante la fase de pruebas del proyecto han sido óptimos. A medida que se iba avanzando en el desarrollo, el número de errores encontrados se reducía enormemente y con el modelo de ciclo de vida seguido su resolución era rápida y con un impacto mínimo.

La actitud colaborativa del cliente ha resultado de grandísima ayuda, su predisposición para revisar los productos generados en cada etapa y para sugerir mejoras y modificaciones han sido determinantes para conseguir un producto mejor y más acorde a sus expectativas.

Los resultados de las pruebas unitarias, de integración y de sistema permiten certificar que el funcionamiento de la aplicación será el correcto sin importar el terminal que utilice el usuario, siempre y cuando cumpla con la versión mínima establecida. La aplicación es capaz de ajustarse a las características de cada dispositivo y de mantener toda su funcionalidad intacta.

Las pruebas de validación han resultado satisfactorias y aportan confianza al equipo para las pruebas finales de aceptación que se realizarán en el futuro. El cliente ha quedado conforme con las muestras que ha podido ir probando y manifiesta su confianza en el producto final.

7 Conclusiones y trabajo futuro

En este último apartado se incluirán comentarios, opiniones y conclusiones personales obtenidas durante la realización del proyecto y posibles mejoras para realizar en el futuro.

Para empezar, me gustaría indicar que los objetivos que nos marcábamos al principio de este proyecto han sido satisfactoriamente cumplidos. Como se esperaba, el alcance del proyecto era demasiado grande, y por eso se decidió desarrollar solo las funcionalidades principales de la aplicación en módulos separados e independientes. Tras la realización del TFG se combinarán todos para dar lugar al primer prototipo completamente funcional del juego. Probablemente lo que mayor trabajo implique, sea la interacción completa del servidor con la aplicación, no por la dificultad si no por el gran número de comprobaciones a realizar. En esta nueva etapa del proyecto se realizarán de nuevo todas las pruebas ya mencionadas, junto con una nueva batería para asegurar que todo sigue funcionando correctamente.

Aunque no se tenían conocimientos previos de programación para Android, ni ningún tipo de dispositivo móvil similar, tras la realización del proyecto considero que soy capaz de desarrollar cualquier tipo de aplicación básica para terminales que utilizan este sistema operativo. Con los conocimientos adquiridos conozco como funciona Android y soy capaz de utilizar y aprender a utilizar las características y funcionalidades que este aporta a sus desarrolladores.

El proceso de aprendizaje comenzó con la lectura de manuales [7] [9] y desarrollando aplicaciones muy básicas para ir comprendiendo como funciona todo el proceso de desarrollo y ejecución de Android. A medida que avanzaba era capaz de implementar productos más complicados, para acabar desarrollando funcionalidades bastante interesantes y comprendiendo el material de apoyo que utilizaría para el desarrollo de nuestra aplicación.

La primera mejora que nos gustaría añadir al prototipo de la aplicación sería modificar la interfaz de usuario. El diseño actual es bastante simple y no conlleva un gran esfuerzo modificarlo, además el cliente ya tiene a un diseñador trabajando para esta aplicación y conseguiríamos que el juego fuera mucho más atractivo. Otra mejora que nos gustaría llevar a cabo es mejorar el servidor, actualmente solo funciona en local y es imperativo que funcione desde cualquier parte con conexión a internet. Primero tendríamos que conseguir que él que tenemos admita conexiones externas y ya más adelante, con el producto preparado para su distribución, tomar la decisión de cómo realizar el hosting.

8 Bibliografía

[1] Girones, Jesús Tomás. *El gran libro de Android*. Marcombo, 2012

[2] Blanco, Paco, et al. *Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone*. *Universidad Politécnica de Madrid*, 2009

[3] Alarcón Barceló, Francisco. *Desarrollo de Aplicaciones para la Plataforma Android: un caso de estudio para el intercambio de libros*. 2014

[4] Ware, Brent, et al. *Open Source Development with LAMP: Using Linux, Apache, MySQL and PHP*. Addison-Wesley Longman Publishing Co., Inc., 2002

[5] Bakken, Stig Sæther, et al. Manual de PHP. *Este manual es© Copyright 1997, 1998, 1999, 2000, 2001 del Grupo de documentación de PHP*. 1997

[6] MYSQL, A. B. MySQL. 2001

[7] Meier, Reto. *A Beginner's Guide to Android*. 2010

[8] Herramienta de consulta Stackoverflow. *http://stackoverflow.com/*

[9] Guías de desarrollo Android. *http://www.talkandroid.com/*

[10] Terminales Android *http://en.wikipedia.org/wiki/Comparison_of_Android_devices*

[11] Jacobson, Ivar; Booch, Grady; Rumbaugh, James. *El proceso unificado de desarrollo de software*. Reading: Addison Wesley, 2000.

[12] Garzas, Javier. Ciclos de vida para gestionar un proyecto software: cascada, espiral, iterativo, incremental o ágil .*http://www.javiergarzas.com/2013/07/ciclos-de-vida-software.htm.* 2013

[13] Información acerca de dispositivos que comparten características, como versión y tamaño. *http://developer.android.com/intl/es/about/dashboards/index.html.* 2013

[14] Wikipedia. Atmosfear. *http://en.wikipedia.org/wiki/Atmosfear_(series)*. 2006

[15] Cheatwell. App-player. *http://www.cheatwell.com/App_Player_Board_Game.htm*

[16] Wikipedia. SQLite. *http://es.wikipedia.org/wiki/SQLite*

[17] Androideity. La importancia del MVC de Android *http://androideity.com/ 2012/05/10/la-importancia-del-mvc-en-android/.* 2012

[18] Blog Sgoliver.net. *Pensamientos varios sobre programación, Android, .NET y Java. Tareas en segundo plano* http://www.sgoliver.net/blog/?p=3099.

Glosario

API	Conjunto de métodos y procedimientos que ofrece una biblioteca.
UI	Interfaz de usuario.
LAMP	Sistema de infraestructura de internet que usa Linux, Apache, MySQL y PHP.
MVC	Modelo vista controlador.
Código QR	Código de rápida respuesta (Quick Response).
SDK	Kit de desarrollo software (Software Development Kit).
APK	Paquete Android (Android Package).
BBDD	Base de datos.
Acoplamiento	Indica el nivel de dependencia entre las unidades software de un sistema informático.
Cohesión	Indica el grado de agrupamiento de unidades del software en otras unidades mayores.
Actividad	Es una clase encargada de representar una única pantalla de la interfaz de usuario.
Intención	Mensajes asíncronos que sirven para enlazar varios componentes independientes, en tiempo de ejecución, sin importar que sean de esta aplicación o de otra.
Fragmento	Representan un comportamiento que se le da a una parte de la interfaz de una actividad.
Listener	Método de cada elemento de la interfaz que se ejecuta cuando dicho elemento es accionado.
Kernel	Núcleo del sistema operativo.
Framework	Estructura conceptual y tecnológica que puede servir de base para la organización y desarrollo de software.
Hosting	Servicio que provee a los usuarios de Internet un sistema para poder almacenar cualquier contenido accesible vía web.

Anexos

A Carga de trabajo

Puesto que este proyecto ha sido realizado por un equipo de dos personas, se ha preferido aclarar las tareas de las que se tenía que responsabilizar cada integrante, para asegurar que se valora el trabajo de cada uno independientemente y de forma justa. A continuación se indica el proyecto que complementa a este:

Desarrollo de la lógica de una aplicación en Android y de la integración de la misma con el entorno físico

Código del proyecto: SIIS_120/1314

A continuación se mostrarán las divisiones de tareas del proyecto y los encargados de la realización de cada una de ellas.

Encargado	**Tarea**
Ambos	Captura y análisis de requisitos
ISSI_119/1314	Diseño e implementación de las bases de datos
ISSI_119/1314	Diseño e implementación la interfaz de usuario.
ISSI_119/1314	Diseño e implementación del servidor.
SIIS_120/1314	Integración de un lector QR
SIIS_120/1314	Integración con el entorno físico
SIIS_120/1314	Diseño de clases
SIIS_120/1314	Diseño e implementación de la lógica del juego
ISSI_119/1314	Implementación de los métodos de acceso a la información
Ambos	Gestión de recursos
Ambos	Pruebas

Tabla 3. Reparto de trabajo

B Entendimiento de la aplicación

En este anexo se añadirán diagramas, códigos y explicaciones que resultaban demasiado extensas para incluir en la memoria y que pretenden mejorar el entendimiento del proyecto.

Ciclo de vida de una actividad

Las actividades atraviesan una serie de estados desde el momento en el que se crean hasta su destrucción. A lo largo de este recorrido se invocan automáticamente un conjunto de métodos. La clase base Activity proporciona versiones por defecto de estos métodos que se pueden sobrescribir. Como ya se explico en el apartado de implementación.

Una actividad puede encontrarse en los siguientes estados:

- **Activa:** Se encuentra en primer plano de la pantalla y tiene el foco del usuario.

- **En pausa:** Sigue visible pero sin el foco. Toda la información de estado se mantiene pero hay otra pantalla delante, como un mensaje.

- **Parada:** La actividad se encuentra detrás de otra y aunque mantiene toda su información podría ser destruida si el sistema necesita más memoria.

Todas las transacciones que se pueden observar en la siguiente imagen van siempre acompañadas por llamadas a los métodos de la clase base Activity:

- **onCreate():** Llamado cuando la actividad se crea por primera vez. Se utiliza habitualmente para crear la interfaz de usuario y proporciona como argumento un objeto de tipo *Bundle* con información acerca del estado anterior de la actividad. Siempre le sigue el método onStart().

- **onStart():** Llamado cuando la actividad se vuelve visible para el usuario. Le sigue inmediatamente onRestoreInstanceState() donde se pueden recuperar los datos guardados previamente en el argumento *Bundle*. Siempre le sigue el método onResume().

- **onResume():** Llamado cuando la actividad empieza a interactuar con el usuario y se suele sobrescribir para ejecutar código que necesite ejecutarse cuando la actividad ya está en primer plano.

- **onPause():** Llamado cuando la aplicación esta reanudando una actividad previa, lanzando una nueva o la actividad está parcialmente visible. Puede ir seguida de un onResume(), un onStop() o en casos extremos un onDestroy(), dependiendo de lo que ocurra después.

- **onStop():** Llamado cuando la actividad deja de ser visible completamente para el usuario porque otra actividad ha sido reanuda o creada y la está cubriendo.

- **onDestroy():** Llamado cuando la actividad se destruye. El método *isFinishing()* permite averiguar si la destrucción se debe a que se invocó el método *finish()* o por que el sistema decidió destruirla porque necesitaba más memoria.

- **onRestart():** Llamado cuando la actividad había estado parada y ha sido reanudada.

Siempre que se sobrescriban estos métodos hay que llamar al mismo método de la superclase o de lo contrario se lanzará una excepción.

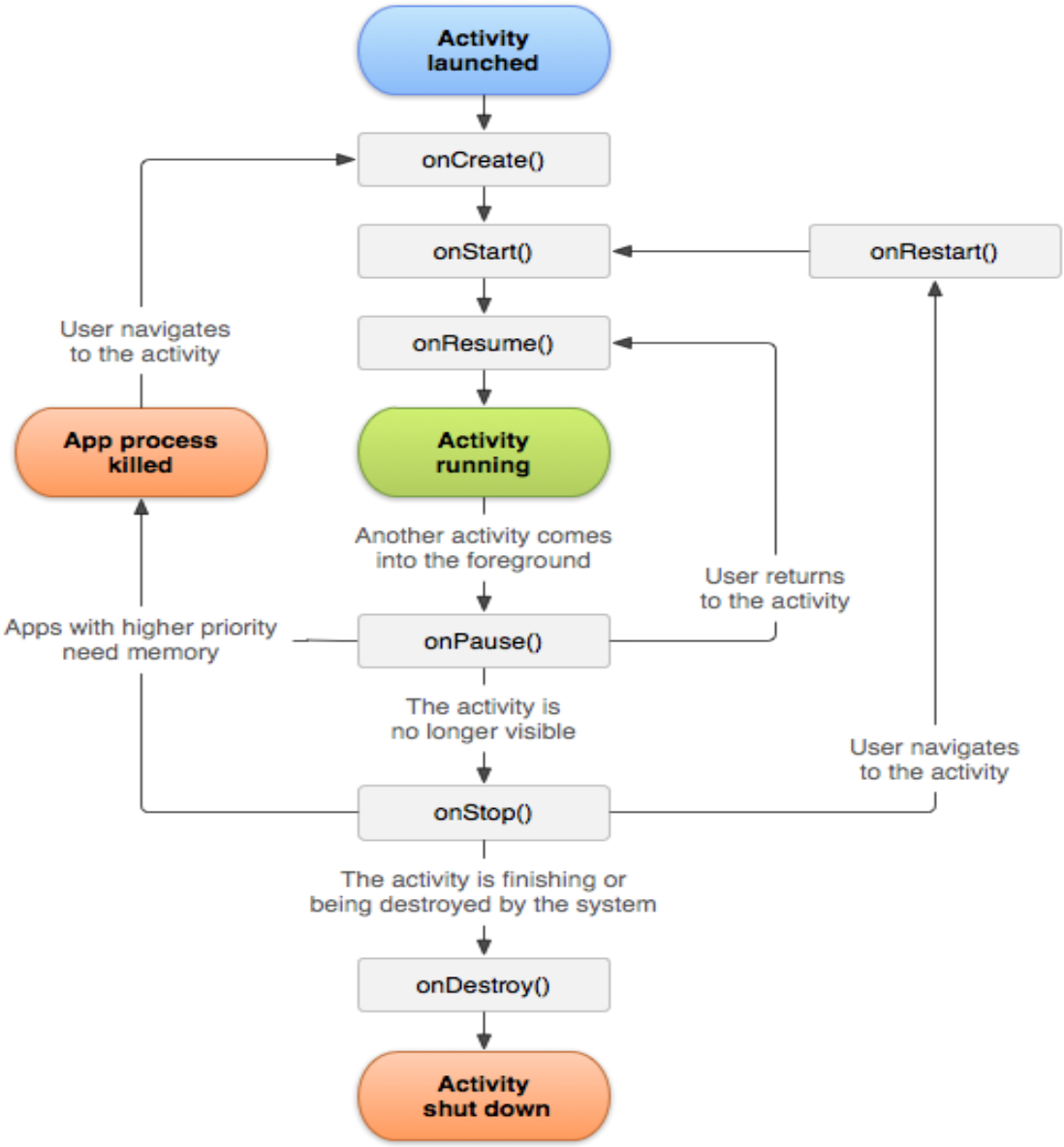


Ilustración 28. Ciclo de vida de una actividad